# CHAPTER 1

**INTRODUCTION**

Android is the most popular operating system (OS) when it comes to mobile platforms. Users prefer Android because of its free and open-source nature with support for many applications. Developers also select Android over the competitive iOS as it is opensource in nature. Security is a crucial aspect of apps. The nature of Android apps makes it difficult to rely on standard, traditional, and dynamic malware analysis systems. Google launched a Google play app security improvement program for providing security services to Google Play app developers to improve the security of their apps. Applications are scanned for potential malware before uploading on Play Store. Google worked on detecting malware and potentially harmful apps for improving security on devices and Play Store using Google Play Protect. To protect user data and passwords, Google has provided a feature for hardware-backed keys. Safe Browsing application programming interface (API) is also present for protection against deceptive websites. While there have been significant developments towards platform security, application development security, and secure Android OS, the apps taking user data can sometimes be malicious.

To prevent the issue of data security and malicious usage of the applications, Android works on the principle of permissions, that is the apps must ask for the permissions required for their applications from the user. The proposed framework provides services to predict the permissions required by an app and instructs the application to prevent malfunction dynamically. The instrumented Android Package (APK) file is installed on the target device. It communicates with the server service at runtime whenever a potentially dangerous API is triggered. The mobile client communicates over the insecure public channel (the Internet). The communicated messages can be read or modified over the network, or the mobile client’s identity can be known to an adversary. Therefore, an anonymous authentication and key agreement scheme are also proposed to protect communication without revealing the client’s identity. Thus, the proposed framework protects user data without affecting the functionality of the application.

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 MODELING AND ENHANCING ANDROID’S PERMISSION SYSTEM**

**Authors: M. Nauman, S. Khan, and X. Zhang (2010)**

This paper makes two main contributions to the body of research on Android security. First, by developing a formal framework for analyzing Android-style security mechanisms, including defining properties desired of those, and verifying whether these properties hold. Second, it by designing and implementing an enforcement system that provides application developers with simple language constructs to specify flexible secrecy and integrity policies, and provably exhibits desirable security properties. The formal framework is composed of an abstract model with several specific instantiations. The model enables to formally define some desired security properties, which can be prove hold on Sorbet but not on Android. Then implement Sorbet on top of Android 2.3.7, test it on a Nexus S phone, and demonstrate its usefulness through a case study.To remain practically relevant, constrain an enforcement system, which can call Sorbet, to be easily retrofittable into Android’s current architecture. The design and implementation of Sorbet improves existing Android permission system in the following aspects:

(1)Formally state the properties about new mechanisms to achieve, and formally prove that the system design supports them.

(2)Enhance Android’s permission system to support coarse-grained secrecy and integrity policies.

(3)Provide more flexible support for fine-grained and scope-limited delegation of permissions.

One of the main goals is to improve understanding of the security properties that desire of Android-like permission systems, and to verify that specific systems are capable of specifying and enforcing desired properties. When pursue this goal by building a generalized, abstract model of the Android permission system, and stating a set of desirable properties in terms of the model. Then develop instantiations of this model both for the current Android permission system and for Sorbet. Based on this formal account, study the properties of the current system. The investigation reveals both design and implementation flaws, which guide the design of Sorbet and prove that Sorbet’s design is sufficient to support the properties that have defined. Android’s permission system only prevents applications that do not have the correct permissions from directly calling a protected component.

This is inadequate to protect against a malicious application that reaches a protected component indirectly, via a chain of calls to innocent applications. To protect against such attacks, Android’s permission system with the ability to specify information flow constraints and explicit declassification permissions, and implement a lightweight calling-context tracking and checking mechanism. Run-time delegation of URI permissions is a key feature in Android, and allows applications to use third-party components to manipulate content that those components normally would not be permitted to access. Android’s implementation of permission delegation is plagued by a number of flaws and questionable design decisions. Sorbet supports more flexible and principled permission delegation and revocation, and allows developers to specify constraints that limit the lifespan and re-delegation scope of the delegated permissions. Developing a mechanism that correctly enforces lifetime and scope constraints turns out to be unexpectedly tricky, due to re-delegation and the dynamic nature of Android applications and components, including application installation and uninstallation, and instantiation and termination of components.

This paper develops a framework for formally analyzing Android-style permission systems, and shows how to enhance Android’s permission system to support rich policies while maintaining convenient, application-centric policy specification. The design of enforcement system satisfies a set of security properties, showed its feasibility by implementing and running it on a Nexus S phone, and demonstrated its usefulness through a case study. The system successfully provides both application and component level protections, but it would need to resort to application-level protection less often if Android’s component-level abstractions were more robust.

**ADVANTAGES**

* provide both application and Component level protection.

**DISADVANTAGES**

* Time consuming.
* No total security.

**2.2 A PRIVACY ENFORCING FRAMEWORK FOR ANDROID APPLICATIONS**

**Authors : R. Neisse, G. Steri, D. Geneiatakis, and I. Nai Fovino (2016)**

The widespread adoption of the Android operating system in a variety type of devices ranging from smart phones to smart TVs, makes it an interesting target for developers of malicious applications. One of the main flaws exploited by these developers is the permissions granting mechanism, which does not allow users to easily understand the privacy implications of the granted permissions. Here, propose an approach to enforce fine-grained usage control privacy policies that enable users to control the access of applications to sensitive resources through application instrumentation. The purpose of this work is to enhance user control on privacy, confidentiality and security of their mobile devices, with regards to application intrusive behaviors. This approach relies on instrumentation techniques and includes a refinement step where high-level resource-centric abstract policies defined by users are automatically refined to enforceable concrete policies. The abstract policies consider the resources being used and not the specific multiple concrete API methods that may allow an app to access the specific sensitive resources. This paper show how approach can be applied in Android applications and discuss performance implications under different scenarios.

Mobile devices are today the primary interface used by end-users to access online services. They are the mean to perform several different operations and they are the repository of a huge amount of sensitive and personal information. Hence, their security and privacy should be the building blocks to enhance trust in digital services. In the mobile world, android is the dominant operating system as it is an open platform that supports different types of hardware devices. However, due to its popularity and its innate design properties, it is also one of the main targets of attackers that want to access to users' private data. Here, analysed different attack vectors leading to private data leakages, and proposed a policy based approach for enhancing users' privacy by empowering them in controlling the access to sensitive resources.

The solution provides flexibility and transparency both to users and apps, by decoupling the specification of security constraint from the enforcement. Performance evaluation outcomes show that the enforcement overhead in terms of processing time is limited, and thus the solution provides a balance between users' privacy and apps “unrestrained” access. The presented approach, at the moment, foresees the enforcement of the policy through app instrumentation. Even if instrumentation can be easily automated to make the operation accessible to the average end-users, recognise that code injection poses some questions related to liability issues of the resulting new “instrumented app”. A far more logical approach would be that of encouraging the mobile app community in developing “privacy by design applications” integrating by default the PEP (policy enforcement point) into their code, giving back to the end-users full control of the behaviour of their mobile devices.

**ADVANTAGES**

* Provides flexibility and transparency both to users and apps.
* Keep customer data secure.
* Process time limited.

**DISADVANTAGES**

* It used mobile processing power.
* Affect the device battery.
* Insecure to device.
* Slower.
* Analysis is done in the user side.

**2.3 ANDROID UNTRUSTED DETECTION WITH PERMISSION BASED SCORING ANALYSIS**

**Authors: H. Shahriar, M. Islam, and V. Clincy (2017)**

Android applications are widely used by millions of users to perform many different activities. However, many applications have been reported to be malware performing activities not matching with their expected behaviors. The existing relevant approaches that identify these applications (malware detection technique) suffer from performance issues where the occurrence of false negatives (FN) remain high. These approaches are not scalable and provides little flexibility to query based on a set of suspicious permission to identify a set of highly relevant known anomalous applications to examine further. This paper proposes a new approach to reduce the number of apps that must be sandboxed in order to determine if they are malicious. First examine the source of applications to identify list of permissions and find a set of highly close and relevant applications from a given set. The identified most relevant application category's permission is then checked to find if there is significant overlapping or not to identify an application as suspected anomalous. Then apply Latent Semantic Indexing (LSI) to identify malware application. In the initial evaluation results suggest that the proposed approach can identify malware applications accurately. The framework uses a formula that will calculate the sensitivity level of the permission and determine if the installed application is untrusted or not.

Permission scoring formula:

{ score = weight\_sum / (weight\_sum+no\_neutral);

}

if (score>= 0.7)

The proposed untrusted detection for android phones that will identify the malicious application that is installed on the device. Untrusted detection was effective and efficient in extracting the permission without decompiling the apk. To get the following permission use getPackageManer(), getPackageInfo(). The permission-scoring formula is effective for evaluating the level of permissions in order to decide if the application is malicious. The Untrusted detection system is effective in providing a solution by detecting the malicious application that can penetrate the android device. Using the permission scoring detection, and it’s satisfies the Untrusted Detection objectives to capture the malware application. Using this anti-malware application, android user will be aware of the applications and its true behaviors.

**ADVANTAGES**

* Detect untrusted application on device.
* Provide security of user data.
* Faster process.

**DISADVANTAGES**

* Cannot use an app without providing permission.

**2.4 TEMPORAL** **PERMISSION** **ANALYSIS** **AND** **ENFORCEMENT** **FRAMEWORK** **FOR** **ANDROID**

**Authors: A. Sadeghi, R. Jabbarvand, N. Ghorbani, H. Bagheri, And S. Malek(2018)**

Permission-induced attacks are among the most critical and frequent issues threatening the security of Android devices. By ignoring the temporal aspects of an attack during the analysis and enforcement, the state-of-the-art approaches aimed at protecting the users against such attacks are prone to have low-coverage in detection and high disruption in prevention of permission-induced attacks. Leveraging temporal logic model checking, TERMINATOR’s analyzer identifies permission-induced threats with respect to dynamic permission states of the apps. At runtime, TERMINATOR’s enforcer selectively leases permissions to apps when the system is in a safe state, and revokes the permissions when the system moves to an unsafe state realizing the identified threats. Detection of several permission-induced attacks, such as those exploiting the TOCTOU (Time of Check to Time of Use) vulnerability in Android requires careful consideration of the order of events.

Hence, the existing conservative prevention techniques, which regardless of the system state enforce security rules permanently, tend to produce plenty of false alarms. As a result, users can be unnecessarily disrupted, even in the absence of material security threats, and prevented from taking full advantage of the apps on their device. Finally, the proposed approaches are mostly realized through modification of either the Android framework or the implementation logic of apps contributes a novel approach and accompanying tool suite, called TERMINATOR, short for Temporal Permission Analysis and enforcement framework for Android. Unlike all prior techniques, TERMINATOR incorporates the notion of time as a first class entity in both detection and prevention of permission-induced attacks introducing the concept of temporal permissions

(1) formulate dynamic aspects of the system over time and reason about the security properties thereof as the system transitions from one state to another (risk detection), and

(2) regulate app permissions at runtime based on the current state of the system (risk prevention). TERMINATOR provides a safe, reliable, yet non-disruptive approach to protect mobile users against permission misuses. Upon receiving a permission request from an app, TERMINATOR evaluates the security posture of the system with respect to the current state of the granted-permission configuration as well as potential threats conservatively identified via the state-of-the-art static analysis tools. If granting the requested permission does not lead to a real security threat given the current state of the system, TERMINATOR leases that permission to the requester. The leased permission is then automatically revoked as soon as a change in the system status is observed that may lead to realization of an identified security threat. TERMINATOR uses TLA+ model checker (TLC) as an analysis engine for temporal permissions. To prevent permission-induced attacks, TERMINATOR relies on the Android’s dynamic permission mechanism without needing to make any modification to the Android framework or the implementation logic of apps

The presented a permission analysis and enforcement framework that, in contrast to the prior work, considers the temporal aspects of permission-induced attacks for their detection and prevention. The framework, called TERMINATOR, is realized in two phases. In the analysis phase, it uses a temporal logic model checker to identify the security risks with respect to dynamic states of the system. In the enforcement phase, it relies on Android’s dynamic permission mechanism to prevent the identified security threats from materializing by regulating the permission configuration of the system TERMINATOR is able to provide an effective, yet non-disruptive, defense against permission-induced attacks. The temporal rules are enforced at the app level, meaning that permissions are leased to the whole app. A more fine-grained temporal rule, that reduces an app’s attack surface further through leasing permissions to a subset of its components, is potentially an interesting avenue of future research.

**ADVANTAGES**

* Effective.
* Non disruptive.
* Highly reliable.

**DISADVANTAGES**

* Not applicable for all apps.
* some app may crash.

**2.5** **ANDROID** **APPLICATION** **RISK** **EVALUATION** **FRAMEWORK** **BASED** **ON** **MINIMUM** **PERMISSION** **SET** **IDENTIFICATION**

**Authors: jianmao xiaoa, shizhan chen , qiang heb, zhiyong fenga, xiao xuea (2020)**

Android utilizes a security mechanism that requires apps to request permission for accessing sensitive user data. However, Android apps tend to be over privileged, that is they often request more permissions than necessary. This raises the security problem of over privilege. To alleviate the over privilege problem, it proposes MPDroid, an approach that combines static analysis and collaborative filtering to identify the minimum permissions for an Android app based on its app description and API usage. Given an app, MPDroid first employs collaborative filtering to identify the initial minimum permissions for the app. Then, through static analysis, the final minimum permissions that an app really needs are identified. Finally, it evaluates the over privilege risk by inspecting the apps extra privileges, that is the unnecessary permissions requested by the app.

The popularity and ubiquitous use of smartphones have greatly fueled the growth of mobile application. Apps are now playing an extremely important role in daily life. Android has long been a major target of malicious apps. One of its major vulnerabilities is the permission mechanism. Android’s permission mechanism requires apps to request permission for accessing sensitive user data, example contacts and SMSs, or certain system features, example camera and Internet access. Thus, the security of Android heavily depends on the effectiveness of this permission mechanism. A major threat is that a malicious app may furtively request extra permissions for accessing users’ sensitive and private data. To minimize this threat, some researchers have designed user-oriented permission prompts to ensure that smartphone users are properly notified of the permissions requested by apps. However, due to the complexity of Android’s permission mechanism, most of these efforts have proven to be ineffective. The main reason is that most users do not fully understand Android’s permissions mechanism. They often simply ignore the prompts and accept apps’ requests for permissions without inspecting the prompts. As a result, apps can easily obtain extra permissions, which increase the risks of user privacy leaks. This is referred to as the over privilege problem.

Here, propose an iteration approach that combines static analysis and collaborative filtering to identify the minimum permission set for the mobile apps. The static analysis is used to achieve the real requested permissions for each mobile app, and a description-minimum permission set iteration algorithm based on collaborative filtering is developed to mine the relationships between the declared functionalities and the minimum requested permissions, so that it can detect the over-requested permissions and identify the high risk applications. Comparing with the previous state-of-the-art method, MPDroid method can effectively identify the abnormal use of permissions and generate better permission recommendation configuration results, there by reducing the problem of mismatch between functionalities and permissions. For the users, only need to have a description information of the app, so that can predict the permission required for the description information according to the model. In addition, this model also can be used to analyze the permissions of existing Android applications and evaluate the risks of the app.

**ADVANTAGES**

* Reducing the problem of mismatch between functionalities and permissions.
* MPDroid method can effectively identify the abnormal use of permissions.
* Generate better permission recommendation configuration results.
* Combines static analysis and collaborative filtering to identify the minimum permission set of Android apps.

**DISADVANTAGES**

* Takes more time.
* User’s permissions are not secure.

**2.6 DANDROID: A MULTI-VIEW DISCRIMINATIVE ADVERSARIAL NETWORK FOR OBFUSCATED ANDROID MALWARE DETECTION**

**Authors: Millara S, McLaughlin N, Martinez del Rincon Miller, & Zhao,Z (2020)**

It present DANdroid, a novel Android malware detection model using a deep learning Discriminative Adversarial Network (DAN) that classifies both obfuscated and un obfuscated apps as either malicious or benign. This method, which can empirically demonstrate is robust against a selection of four prevalent and real-world obfuscation techniques, makes three contributions. Firstly, an innovative application of discriminative adversarial learning results in malware feature representations with a strong degree of resilience to the four obfuscation techniques. Secondly, the use of three feature sets, raw opcodes, permissions and API calls, that are combined in a multi-view deep learning architecture to increase this obfuscation resilience. Thirdly, it demonstrate the potential of the model to generalize over rare and future obfuscation methods not seen in training.

Traditional machine learning (ML) systems using statistical methods and feature sets hand-crafted by malware experts can be slow to react to changing and new threats. Unless this laborious manual engineering and ranking of features continually takes place over time, these detection methods risk only being fully effective in the short-term. In addition, relatively few research publications have advanced the classification of obfuscated Android malware. This presents an opportunity for new techniques that do not need expert malware domain insight to generate or rank features, and can display a level of resilience to obfuscation. Obfuscation is a challenging problem for current detection systems, with Android malware authors regularly using techniques like encryption, reflection and reference renaming. These aim to disguise and camouflage malicious functionality in an app, tricking a model into classifying it as benign. They propose several approaches to addressing the obfuscation problem. Firstly, present a deep learning Discriminative Adversarial Network (DAN) with two cost functions, one to minimise classification error for malware, and the other to maximise classification error for obfuscation, ensuring the internal features learned are useful for malware detection whilst simultaneously being ignorant of obfuscation caused by four commonly used techniques. The DAN employs the adversarial learning aspect of a GAN but instead of training a generative model, train two discriminators, one for malware and another for obfuscation. This learning algorithm is inspired by previous work from other fields in domain adaptation and feature disentanglement, with obfuscation considered a form of bias to remove from the learning process. Secondly, adopt a multi-view learning approach that uses obfuscation-resilient feature sets of permissions and API calls, in addition to raw opcode sequences that are extracted directly from each Android app. The model’s resilience to obfuscation is tested against four specific techniques applied individually, and in combination - class encryption, API calls obfuscation, string encryption and resource encryption. The evidence the features learned by a multi-view DAN model are less affected by obfuscation compared to without using DAN. Lastly, through considering obfuscation as a form of data augmentation, and augment the original training data with the obfuscated versions of the same malicious and benign apps, annotating each sample with two labels; either malicious or benign, and either obfuscated or unobfuscated. A novel deep learning approach that performs strongly in detecting obfuscated and unobfuscated Android malware is presented, which will name as DANdroid. Results compare favorably with the state-of-the-art.

Using a multi-view DAN architecture remove obfuscation bias from the learning process via negation of the obfuscation cost in a bespoke adversarial cost function. A balanced, augmented dataset of both obfuscated and unobfuscated malicious and benign apps, combined with a multi-view architecture learning from opcodes, permissions and API calls, provides more confidence in the discriminative power of learned features when compared to state-of-the-art methods that attempt to deal with obfuscation. Further analysis proves the model is resilient to the four selected obfuscation methods used in training, and also demonstrate its potential to generalize over rare and future obfuscation methods not seen in training. The model is highly effective in detecting malicious samples obfuscated with multiple techniques, including those which thirteen commercial anti-malware engines struggled with.

**ADVANTAGES**

* Excellent performance despite the presence of four selected obfuscation techniques.
* Classifies both obfuscated and unobfuscated apps.

**DISADVANTAGES**

* Takes more time.

**2.7 APPLICATION SECURITY FRAMEWORK FOR MOBILE APP DEVELOPMENT IN ENTERPRISE SETUP**

**Authors: Li, R, Diao, W, Li, Z, Yang, S, Li, S, & Guo, S (2021)**

Enterprise Mobility has been increasing the reach over the years. Initially Mobile devices were adopted as consumer devices. However, the enterprises world over have rightly taken the leap and started using the ubiquitous technology for managing its employees as well as to reach out to the customers. While a significant focus have been put on network security, this paper discusses on the approach that can be taken at Mobile application layer, which would reduce the risk to the enterprises. While Mobility has increased its reach rapidly as a consumer electronic device, smart enterprises have been using the technology to its benefit. While the major focus areas have been in device, application & network side, the importance of security is also emerging now. CIOs have been approaching the Enterprise Mobility initiatives mostly from Application side. Due to its lack of focus, this paper attempts to discuss a Mobile App Security Framework. This can help any Enterprise which is embarking on its Enterprise Mobility journey to reduce the risks related to Enterprise Data, prospective customers & demographic details.

Mobile operating systems, like Android, provide a system security model as part of its stack. This prevents many risks by virtue of Operating System level controls. However further planning needs to be done to take care of issues like data loss, Intellectual Property violation etc. Considering the application framework that has been mentioned in the paper, it is important for any organization embarking into Enterprise Mobility journey to have a clearly defined coding standard. This standardization of approach towards coding standard and compliance to the same can help mitigate the security related issues in Mobile applications to a large extent and make the Enterprise Mobility endeavor successful.

**ADVANTAGES**

* It reduce risk from both internal and third party sources.
* Keep customers data secure.

**DIADVANTAGES**

* Poor encryption.
* Lack of input validation.

**2.8 TARGETED AND DEPTH-FIRST EXPLORATION FOR SYSTEMATIC TESTING OF ANDROID APPS.**

**Authors: Tanzirul Azim, Iulian Neamtiu(2021)**

Systematic exploration of Android apps is an enabler for a variety of app analysis and testing tasks. Performing the exploration while apps run on actual phones is essential for exploring the full range of app capabilities. However, exploring real-world apps on real phones is challenging due to non-determinism, non-standard control flow, scalability and overhead constraints. Prior approaches for automated exploration of Android apps have run apps in an emulator or focused on small apps whose source code was available. To address these problems, A 3E, an approach and tool that allows substantial Android apps to be explored systematically while running on actual phones, yet without requiring access to the app’s source code. The key insight sapproach is to use a static, taint-style, dataflow analysis on the app bytecode in a novel way, to construct a high-level control flow graph that captures legal transitions among activities (app screens). Then use this graph to develop an exploration strategy named Targeted Exploration that permits fast, direct exploration of activities, including activities that would be difficult to reach during normal use, and also developed a strategy named Depth-first Exploration that mimics user actions for exploring activities and their constituents in a slower, but more systematic way. To measure the effectiveness of the techniques, use two metrics: activity coverage (number of screens explored) and method coverage.

This approach is focused on improving coverage at two granularity levels: activity (high-level) and method (low-level). Activities are the main parts of Android apps an activity roughly corresponds to a different screen or window in traditional GUI-based applications. Increasing activity coverage means, roughly, exploring more screens. For method coverage focus on covering app methods, as available in the Dalvik bytecode (compiled from Java), that runs on the Dalvik VM on an actual phone; an activity’s implementation usually consists of many methods, so by improving method coverage allow the functionality associated with each activity to be systematically explored and tested. That provide an overview of the Android platform and apps, define the graphs that help drive the approach, and provide definitions for coverage metrics. To understand the level of exploration attained by Android app users in practice, and performed a user study and measured coverage during regular interaction. Then have presented A 3E, an approach and tool that allow Android apps to be explored systematically. Then performed a user study that has revealed that users tend to explore just a small set of features when interacting with Android apps.

Then have introduced Targeted Exploration, a novel technique that leverages static taint analysis to facilitate fast yet effective exploration of Android app activities, and also introduced Depth-first Exploration, a technique that does not use static analysis, but instead performs a thorough GUI exploration which results in increased method coverage. this approach has the advantage of permitting exploration without requiring the app source code.

**ADVANTAGES**

* Permitting exploration without requiring the app source code.
* Android apps to be explored systematically.

**DISADVANTAGES**

* Takes more time.

**CHAPTER 3**

**EXISTING SYSTEM**

**3.1 AVG SECURITY AND VIRUS CLEANER**

Experience supreme android antivirus and privacy pro. Mobile security for Android to help protect you from harmfull viruses and malware. Keep your personal data safe with App lock,photo Vault,Wi fi security Scan,hack Alert,Malware security and app Permissions advisor. This app uses Device Administrator permission Discover potential privacy issues.This app uses Accessibility permission to protect visually impared and other user against phishing attacks and malicious websites.

**ADVANTAGES**

* If there is any issues it get fixed soon and solve all problems.
* Fully protected with real time scanning,daily cleaning,and all of the other components.

**DISADVANTAGES**

* This App is paid.
* Takes more Time.

**3.2 BITEFENDER MOBILE SECURITY.**

Bitefender mobile security and antivirus is the most powerful antivirus app for android. It protects your smartphone and tablets against viruses, malware and online threats and keeps your private information secure from hackers**.** Antivirus Protection – keeps your Android device safe from all new and existing online threat Virus  
 Web Protection – secures your Android device against dangerous, phishing and fraudulent sites for the most popular browsers. Autopilot–makes security recommendations based on your mobile device usage Account Privacy - monitors if your email accounts were involved in any data breach VPN - makes your IP anonymous & unlocks geo-restricted content. App Lock - protects your sensitive and private mobile apps with a PIN code or fingerprint Anti-Theft - protects your personal information in case your Android device is stolen or lost.  Reports - sends insights about your weekly activity and security incidents that were prevented. WearON - extends the protection offered by Bitdefender Mobile Security & Antivirus to your smart watch Bitdefender Mobile Security & Antivirus keeps your Android phone and tablet protected from all online threats. It acts as a virus cleaner and malware removal tool.

**ADVANTAGES**

* Power full scan against mobile threat.
* It act as an virus cleaner.
* Keeps private information secure from hackers.

**DISADVANTAGES**

* Difficulty to use.
* Slower.

**3.3 REVO APP PERMISSION MANAGER**

Easy and useful app for managing all permissions on your smart phone . Easy and useful app for managing all permissions used in background on your smartphone. Be always informed about what permissions you give access to, and what information the app is collecting. Lists installed apps according to their risk priority (High Risk, Medium Risk, Low Risk, No Risk). Search bar for easily finding of an app. Detailed description of permission privileges the app enjoys Easily remove any risky permissions .Sort apps by high and low risk level.Permission viewer: Filter easily apps by permission group categories /Calendar, Camera, Contacts, Location, Microphone, SMS, Storage, Phone, Sensors.Now the Professional version of Revo App Permission Manager provides more control of everything. Take a look and see the risky permissions you are giving to some applications without knowing.

* Allows applications to discover and pair Bluetooth devices.
* Allows applications to change Wi-Fi connectivity state.
* Allows applications to open network sockets.
* Allows an application to modify global audio settings.
* Allows applications to perform I/O operations over NFC.
* Allows a companion app to run in the background.
* Allows using the device\'s IR transmitter, if available.
* Allows an app to use device supported biometric modalities.
* Allows access to the vibrator.
* Inn-app purchase and subscription.
* Allows an application to send SMS messages.
* Allows an application to record audio.
* Allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call.
* Access to precise location.
* privacy and security is one of the most important things in terms of control. and knowledge.

Revo App Permission Manager gives you the power to be always informed and to have the ability to control without you even knowing with your most sensitive information, like location, contacts, camera and even microphone. The main aim of the app permission is to protect the privacy of an Android user. Android apps must request permission to access sensitive user data such as Contacts and SMS, as well as certain system features such as Camera and Internet. Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request. It is always a good idea to be aware what permissions the apps on your Android device are using in the background, especially if you don't know for their purpose.

Once you installed Revo App Permission Manager and start the application, you will see all risky categories in terms of potential high, medium or low risk. You can easily find the desired application in the search field or you can take a look on all apps through the risk categories. By selecting the app you are interested in you can see all the information about the current permissions the app is using and easily stop them.

**ADVANTAGES**

* Easily remove any risky permissions.
* protect the privacy of an Android user.
* Easy and useful app for managing all permissions on your smartphone.

**DISADVANTAGES**

* Active permission cannot be shown correctily.
* Removed permissions cannot be shown correctily.

**3.4 APP PERMISSION MANAGER**

A complete solution for mobile security and privacy assessment.The permissions used by all installed apps in your device and it allows you to revoke permissions on one tap. This App uses “Accessibly Service ” to perform revoke app permissions. Get Notification of risk associated with newly installed app.

**App Features:**

You can check and allow/revoke following permissions

1. Installed apps permissions.
2. Special permissions for apps.
3. System apps permissions.
4. Group permissions.
5. For Android Version 6.0 and above.
6. Lists installed apps by their severity (High Risk, Medium Risk, Low Risk)
7. Click on any app and you will get details of all permissions used by it.
8. If you want to remove any risky permission, turn off button and click on APPLY CHANGES button.

**ADVANTAGES**

* Get Notification of risk associated with newly installed app.
* Easy to use.

**DISADVANTAGES**

* Slower.

**3.5 BOUNCER**

A better way to manage app permissions. Bouncer gives you the ability to grant permissions temporarily. Want to tag a location or take a photo, but don't want that app to be able to use the camera or get your location whenever it wants? bouncer gives you exactly that. As soon as you exit the app, Bouncer will automatically remove the permission for you in an instant so you can get back to doing what you do best, without having to worry about apps invading your privacy and wasting your battery. Bouncer is designed for one-time permissions, not for permissions where you only want the app to have them in the foreground.

Bouncer can be used for both purposes but as you use apps throughout the day they will be in the foreground a lot of the time. Increased security, privacy and battery life. Never have to worry what apps are doing in the background. No complicated setup needed (no root or adb) Some devices work better with Bouncer than others due to different restrictions. Some devices like Nokia devices may prevent Bouncer from running because of its aggressive battery management.

**ADVANTAGES**

* Ability to grant permissions temporarily.
* Easy to use.

**DISADVANTAGES**

* This app is paided.

**3.6 PERMISSION CHECK PLUGIN**

Permission Check Plugin Detects permissions the installed apps require, so that you could take action to safeguard your privacy. With this plugin installed, you are able to see clearly the permission apps require without Checking them one at a time. You could press the item that requires the permission critical to you, view more details about it, or uninstall it. About All-In-One Toolbox. All-In-One Toolbox is the most comprehensive tool app to optimize your Android in optimal speed. Trusted by Millions of users world wide.

**App Features**

* Clean all process.
* cache, temp files, empty folders, residual files, app leftovers, clipboard data, SMS & call logs, saved password, search history, large files.
* Task Killer & Memory (RAM) Booster.
* Kill background running tasks whenever device slows down, lags or freezes to boost speed.
* Pre-Installed & User Installed App Manager.
* Manage all apps.
* Root user can customize auto-start apps, move app to SD card, batch install&uninstall apps silently, uninstall build-in system apps.
* Useful Plugins.

1. App Advertisement Detector

2. App Locker

3. Auto Clean & Boost Tasks

4. Game Booster

5. App Permission Check

6. Compass

7. Flashlight

8. QR & Bar code Scanner

9. Quick System Settings

10. Sensor Box

11. Quick Volume Settings

12. Pro Key

**ADVANTAGES**

* Detects permissions the installed apps require.

**DISADVANTAGES**

* Takes More Time For Working.
* Slower

**PROBLEM DEFINITION**

It is found that the existing methods do very little to determine how to stop the applications from being malicious and still use it; almost all of them used mobile processing power for the framework and did not learn with time. One of the works in which permissions were being revoked after use, also does not prevent the application from using user data at runtime. This study addresses all these issues by providing a system that analyses an application’s functionality permission by permission and prevents them from using user data potentially for malicious purposes. The results that are obtained from the server during analysis are processed and stored for all new applications.

**CHAPTER 4**

**PROPOSED SYSTEM**

The system propose an end-to-end framework to ensure the proper functioning of the app along with user privacy protection. A background Data Protection Service, which is installed on the user's phone, runs to capture dangerous API calls on the runtime. It returns garbage data to be sent back to the malicious app. Dedicated server analyze and instrument apps that the user is using for making it compatible with the proposed service. The proposed approach uses two algorithms to determine malicious permissions asked by the apps, which needs some initial dataset to work on.

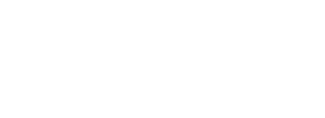
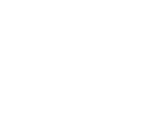


Fig 4.1: Flowchart Of The Proposed Framework.

The following two components are presented in the proposed framework:

**A. ANALYSIS AND INSTRUMENTATION OF THE APK**

The analysis of the app is done using the resources and permissions the app requires. Analysis of permissions against the API calls and the utility of the application is done. The permission recommendations are used to predict whether the application is demanding extra permissions for stealing user data. For that purpose, the algorithms collaborative filtering and frequent permission set mining are used on permissions. The training set is based on the data collected for various applications in various categories (as described in the later section). The result formed by the intersection of the results from both the algorithms individually containing unsafe and extra permissions is sent to the instrumentation engine, which instruments the APK to support functionality to call the Data Protection Service. Both algorithms work by classifying the permissions of app based on the category that lies in in the play store, and the acquired dataset.

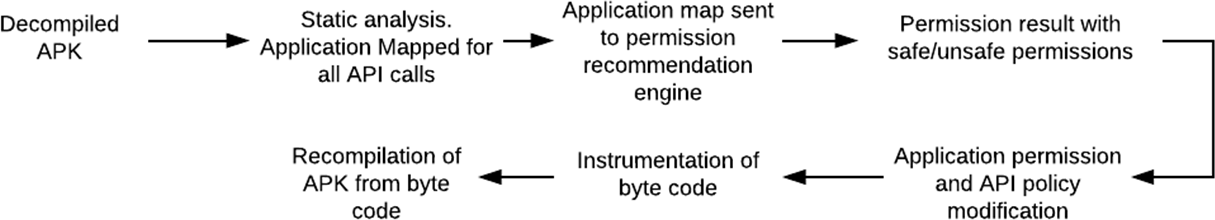


Fig 4.2 : The Flow Instrumentation Engine

**B. DATA PROTECTION SERVICE**

To prevent user privacy, the proposed service runs in the background on the user's phone andsends the APK file of the installed app to the server for analysis and instrumentation. It facilitates the installation of the instrumented APK for the user. Finally, when the application starts running, it provides garbage data to the app whenever an identified malicious call to API is made. Garbage data is produced using the broadcast receiver. The Android framework has a facility to allow users to register for events using a broadcast receiver according to the lifetime i.e., statistically and dynamically. In the case of dynamic, the lifetime depends upon Context.registerReceiver() and Context.unregisterReceiver() on the app component. In the case of static, a receiver is specified in the AndroidManifest.xml and has an identical lifetime to the app. The receiver utilizes a callback approach i.e., BroadcastReceiver.onReceive(), to override SDK calls.

The following approaches on which the proposed framework works are described in detail:

1. **Data Collection**

The data collection approach is divided into two parts. The initial data collection is done by developing and circulating one data collection app. This app is downloaded by roughly 300 users through which information about 1000 unique applications is collected. Thereafter, the data of asked permissions and the permissions provided by the user are extracted, and the half probability rule is used to determine whether the permission is necessary or malicious. Afterward, whenever the algorithms run on the server, unique app data is added to the database, which would help to increase the dataset and help the proposed framework learn with time.

1. **Analysis and Instrumentation**

The engine on the server runs to analyze and instrument APKs. It begins with the decompilation of the app using Apktool which is used to reverse engineer Android apps. It decompiles app into Smali code that is, the assembly code that runs on the Dalvik Virtual Machine (Android's Java Virtual Machine). The decompiled code goes through the following stages in the parsing and instrumentation engine. The application is then repackaged using Apktool. It is installed on the user's Android phone by the Data Protection Service.

**1) STATIC ANALYSIS**

The original Smali code is given as input to the proposed engine. Static analysis is performed by parsing Smali code files. A map of app is created, which shows the file name and the methods included in the file, class names, and API calls. A clear map is created for reference in instrumentation in the later stages. The manifest file is parsed separately. All permissions are extracted from the file, and dangerous permissions from the superset are kept for analysis. A python-based parser handles the manifest file and Smali code; it traverses the directory of the decompiled APK and maintains a record of class names, method names, and API Calls for each file. A map is created for all method calls. During enforcement of redefined permission policies, the map is used to locate the functions and files to be instrumented.

**2) PERMISSION ANALYSIS**

As discussed earlier, two approaches were used to identify the permissions required by an app: collaborative filtering and frequent permission set mining. The training set consists of apps of all categories as available in the Google play store.

1. **Collaborative Filtering**

Collaborative filtering is one of the commonly used techniques in recommender systems. It utilizes the information contained in a group to recommend information on a new entity related to the group. It is based on the idea that entities that share certain evaluation criteria of certain items in the past are likely to agree again in the future. Feature vectors are used in this method to represent the items in the entity that goes through the evaluation process of finding a similarity score.

**(i)Finding the feature vector in the proposed engine**

The app permissions are used as feature vectors for the collaborative filtering engine. Permissions of an app are extracted in a vector as 𝑉 =< 𝑃1 , 𝑃2 , … , 𝑃𝑛 >, where 𝑃𝑖 can take values from the set [0,1] depending upon whether the app takes that permission. Feature vectors from the apps in the training data set are taken. The engine first extracts the apps in the same category as that of the test app. It then extracts all the permissions in the feature vectors for filtering and recommendation. 𝐴𝑖 = {𝐴𝑃1 , 𝐴𝑃2 , … , 𝐴𝑃𝑛 }, where 𝐴𝑃𝑖 takes the value from the set [0,1].

**(ii) Evaluation of similarity**

A similarity score is a measure of how closely related two entities are. The similarity is calculated for the app with all the apps in the same category using the Jaccard similarity score as (𝐴 , 𝐴𝑡 ) = 𝐹11 (𝐹01+𝐹10+𝐹11) 𝐴𝑖 is the app from the data set in the same category. 𝐴𝑡 is the test app. 𝐹11 is the frequency of matches in the permissions between 𝐴𝑖 and . 𝐹01 is the frequency of permissions 1 in the case of 𝐴𝑖 and 0 in the case of . 𝐹10 is the frequency of permissions 1 in the case of 𝐴𝑡 and 0 in the case of .

**(iii) Recommendation of Permissions**

A recommendation score is generated for each permission request by the app. It can be calculated using 𝑅𝑆𝑐𝑜𝑟𝑒𝐶𝐹 ( ) = ∑𝑆(𝐴𝑖 , 𝐴𝑡 ) Here, majority voting is considered, with the voting's weight proportional to the similarity score generated above. The generated 𝑅𝑆𝑐𝑜𝑟𝑒 is normalized. Depending on the score, the permission is marked as safe if recommended for the app to be used; otherwise, it is marked unsafe. The step-by-step flow of the permission segregator is presented in Algo. 1.

**Algorithm 1: Permission segregator**

1. 𝐽𝑎𝑐𝑐𝑎𝑟𝑑𝑆( )

2. for 𝑖 in range(len(app)):do

3. check the vector scores 11, 10 and 01

4. if " 11" then

5. → num++, denum++

6. elif "01" or "10" t hen

7. → denum++

8. return num / denum

9. 𝐹𝑖𝑛𝑑𝑆𝑢𝑝𝑝𝑜𝑟𝑡𝐹𝑜𝑟𝑃𝑒𝑟𝑚𝑖𝑠𝑠𝑖𝑜𝑛𝑠 ( ) {

10. → every asked permission is analyzed based on category present a score is calculated and provided using Jaccard similarity for permission set

11. return permissionScore[ ] }

12. perms[ ] = FindSupportForPermissions ( )

13. for 𝑖 in range (len (perms) ):do

14. if (perms[ 𝑖 ] > threshold ) then

15. self.safePermissions.append ( self.permsAt[ 𝑖 ])

16. else

17. self.unsafePermissions.append (self.permsAt[ 𝑖 ])

**b) Frequent Permission Set Mining**

The second recommendation algorithm is based on predicting permission pair values that occur together. Relationships and patterns of the permissions requested simultaneously are studied. Based on the relationship that two permissions share, permissions are recommended for an application. Preliminary support calculation in predicting an event is based on frequency. Support is used to discover relationships among entities. Suppose an event (event 𝐵) taken from a dataset of 𝑁 events occurs 𝑓 times (frequency of event 𝐵). The support of event B is 𝑆𝑢(𝐵) = 𝐹𝑟𝑒𝑞𝑢𝑒𝑛𝑐𝑦(𝐵) 𝑁. The permissions of the proposed test app 𝐴𝑡 are extracted in 𝑉 =< 𝑃1 , 𝑃2 , … , 𝑃𝑛 > where 𝑃𝑖 can take values from the set depending upon whether the app takes the permission. Vectors of training data apps containing their permissions as 𝐴𝑖 = {𝐴𝑃1 , 𝐴𝑃2 , … , 𝐴𝑃𝑛 }, where 𝐴𝑃𝑖 takes the values from the set.

**(i)Training of the Proposed Model**

Applications belonging to the same category follow some pattern of frequently co-occurring permissions as:

𝐴𝑖 =< 𝑃1 , 𝑃3 , 𝑃5 , 𝑃6 >

𝐴𝑖+1 =< 𝑃1 , 𝑃2 , 𝑃4 , 𝑃3 , 𝑃7 >

𝐴𝑖+2 =< 𝑃1 , 𝑃3 , 𝑃8 , 𝑃10 >

𝐴𝑖+3 =< 𝑃5 , 𝑃4 , 𝑃7 >

Taking in pairs < , 𝑃𝑗 >, the support of the co-occurring permission pair is calculated as: 𝑆𝑢𝑝(< 𝑃𝑖 , 𝑃𝑗 >) = 𝐹𝑟𝑒𝑞() 𝑁

where 𝑁 is the total number of applications in the category 𝐹𝑟𝑒𝑞(< 𝑃𝑖 , 𝑃𝑗 >) is the frequency of the pair < 𝑃𝑖 , 𝑃𝑗 > when it is requested together.

**(ii) Recommendation of permissions**

The support calculated for all pairs is analyzed.

𝑖𝑓 𝑆𝑢𝑝(< 𝑃𝑖 , 𝑃𝑗 >) > 𝑡 𝑡ℎ𝑒𝑛

𝑅𝑒𝑐𝑜𝑚𝑚𝑒(< 𝑃𝑖 , 𝑃𝑗 >) = 1

Here, 𝑡 defines the threshold value. Permission pairs having a support value higher than threshold are marked safe and recommended. Rest is marked unsafe 𝑅𝑒𝑐𝑜𝑚𝑚𝑒(< 𝑃𝑖 , 𝑃𝑗 >) = 0. After calculating safe permissions from each recommender, the intersection of the resulting permission sets is the final permission set that will be used for further processing. Algorithm. 2 presents the various steps of the permission miner.

**Algorithm 2: Permission Miner**

1. Permissions are numbered from 0 to 𝑛 − 1.

2. Data is then read to identify patterns.

3. for 𝑖 = 0 to 𝑛 − 1 do

4. Identify and store the dangerous permissions

5. let there be 𝑘 permissions

6. for 𝑖 = 1 to 𝑘 do

7. for 𝑗 = i+1 to k do

8. Support between permissions 𝑖 and 𝑗 is calculated and stored { Support is calculated using: for 𝑖, row in Rows do if (values in two col same) then count ++ }

9. Max, Min, and Avg support are calculated for the 𝑘 permission set. for 𝑖 in 1 to 𝑘 do Max = Max > support[ 𝑖 ] ? Max: support[ 𝑖 ] Min = Min < support[ 𝑖 ] ? Min : support[ 𝑖 ] sum = sum + support[ 𝑖 ] Avg = sum / 𝑘

10. for key, value in the stored list do

11. if (key > Avg ) then

12. Permission required

13. else

14. Permission not required

**C. INSTRUMENTATION**

In this study, our primary focus is on dangerous permissions. The permissions suggested by the permission recommendation engine are fed into the instrumentation engine. The policies suggested by the permission recommendation engine are marked safe. The instrumentation engine modifies the policies marked as unsafe. Smali code is instrumented to facilitate communication with the background service at runtime. Hence, through instrumentation, the communication between the malicious detected app and the background service through broadcast receivers is enabled. Background processes are utilized as services on Android. These processes do not provide graphical components and are implemented for background activities for a given program. All services utilized by an app must be added in the manifest. Permissions that are marked unsafe are injected with the piece of code invoked by the required services using broadcast receivers. All the unsafe policies are instrumented within the app then repackaged using Apktool.

**CHAPTER 5**

**SYSTEM REQUIRMENTS**

**Hardware Specification**

* Processor : i5 or i7 (i7 is better)
* RAM : 12GB (Minimum)
* Hard Disk : 500GB or above
* Keyboard

**Software Specification**

* Tool : Python IDLE,
* Python : version3
* Operating System : Windows 7

**Technology : python**

Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigm), including [structured](https://en.wikipedia.org/wiki/Structured_programming) (particularly [procedural](https://en.wikipedia.org/wiki/Procedural_programming)), [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) and [functional programming](https://en.wikipedia.org/wiki/Functional_programming). It is often described as a "batteries included" language due to its comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library). Python uses [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_typing) and a combination of [reference counting](https://en.wikipedia.org/wiki/Reference_counting) and a cycle-detecting garbage collector for [memory management](https://en.wikipedia.org/wiki/Memory_management). It uses dynamic [name resolution](https://en.wikipedia.org/wiki/Name_resolution_(programming_languages)) ([late binding](https://en.wikipedia.org/wiki/Late_binding)), which binds method and variable names during program execution. Its design offers some support for functional programming. It has  filter, map and reduce functions; [list comprehensions](https://en.wikipedia.org/wiki/List_comprehension), [dictionaries](https://en.wikipedia.org/wiki/Associative_array), sets, and [generator](https://en.wikipedia.org/wiki/Generator_(computer_programming)) expressions. The standard library has two modules that implement functional tools borrowed from [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language)) and [Standard ML](https://en.wikipedia.org/wiki/Standard_ML).

**Front end : Tkinter**

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter is the de facto way in Python to create [Graphical User interfaces (GUIs)](https://www.activestate.com/blog/top-10-python-gui-frameworks-compared/) and is included in all standard Python Distributions. In fact, it’s the only framework built into the Python standard library. This Python framework provides an interface to the Tk toolkit and works as a thin object-oriented layer on top of Tk. The Tk toolkit is a cross-platform collection of  ‘graphical control elements’, aka widgets, for building application interfaces. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit. Creating a GUI application using Tkinter is an easy task. Tkinter calls are translated into Tcl commands, which are fed to this embedded interpreter, thus making it possible to mix Python and Tcl in a single application. There are several popular GUI library alternatives available, such as [wxPython](https://en.wikipedia.org/wiki/WxPython), [PyQt](https://en.wikipedia.org/wiki/PyQt), [PySide](https://en.wikipedia.org/wiki/PySide), [Pygame](https://en.wikipedia.org/wiki/Pygame), [Pyglet](https://en.wikipedia.org/wiki/Pyglet), and [PyGTK](https://en.wikipedia.org/wiki/PyGTK).

**Back end : python**

Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is dynamically-typed and garbage-collected.

**CHAPTER 6**

**ARCHITECTURAL DESIGN**

**Dataset preparation**

Save permission and catagory

Dataset

Extract permission

Input APK

IIINPUT APK

Input APKII

**Testing stage**

Datasetd

calculate recommendation score

Input APK

Select permission

Compare with dataset

Extract Permission

Apply frequent set mining

Select permissions

Change permission dataata

Decompile apk

TestTest apk

Fig 6.1: Architectural design

**Dataset preparation**

For creating the dataset add the input apk file of a particular applications. Then extract the permissions required for the apps. Save the permissions and their category. Then all these informations are stored on the dataset.

**Testing stage**

For testing stage, input the apk file for an app. Then extract the permissions required for the app. Compare it with the dataset prepared. Apply frequent set mining algorithm and calculate the recommendation score. After the score is calculated select the permissions. Then decompile the apk file. After that change the permissions that are not required for the application. Then test the apk file.

**DATAFLOW DIAGRAM**

**Level 0 :**

Fig 6.2: level 0 DFD

It is also known as a context diagram. It’s designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as a single bubble with input and output data indicated by incoming/outgoing arrows. First the user enter the apk file to the system then request permission is given to the dataset. After that the dataset give the response back to the system and then the system give response back to the user.

**Level 1:**

user

Fig 6.3: level 1 DFD

In 1-level DFD, the context diagram is decomposed into multiple bubbles/processes. In this level, the main functions of the system and breakdown the high-level process of 0-level DFD into subprocesses. In level 1 DFD the user input the apk file. Then permissions are extracted. Then by using collaborative filtering and frequent set mining algorithm select the suitable permissions required. Then the apk permissions are changed. Then the apk is tested.

**CHAPTER 7**

**CONCLUSION**

The smartphone market has grown extensively in recent years and has become a repository for users’ private data making the security of the device a big challenge. As technology advances, the risk of data breaches and invasion of privacy increases. Various research approaches were presented to identify the malicious behavior of Android applications. A privacy-preserving secure framework was proposed to prevent the applications from stealing user data by restricting all unnecessary permissions using instrumentation and re-packaging of the application. These permissions were recognized by predicting the permissions required by a given Android app by using collaborative filtering and frequent permission set mining algorithms. Thus, the proposed model interacts with the target app and modifies the permission data inside. A layer of security was added in proposed framework to prevent attackers from intercepting communications. Therefore, the proposed framework is more secure and efficient than the competitive models.

However, this approach may achieve poor results for sealed protected applications that generally come under the category of finance/ payments as these applications come with additional security. Hence, these apps cannot be installed after they have been instrumented. In the future, the framework can be modified to make it resilient to the additional securities/ protections in the applications.

**REFERENCES**

**[1] Bharavi Mishra, Aastha Agarwal, Ayush Goel, Aman Ahamed Ansarai***”Privacy protection framework for android* “*Privacy protection framework for android* ”

[**2**].***Android Security 2018 Year in Review 2*.** Accessed: Oct. 19, 2020

**[3].Biggest App Stores in the World 2020** *| Statista*. Accessed: Nov. 22, 2020. [Online]. Available: [https://www.statista.com/statistics/276623/number](http://www.statista.com/statistics/276623/number-)- of-apps-available-in-leading-app-stores/

**[4].K. W. Y. Au, Y. F. Zhou, Z. Huang, P. Gill, and D. Lie**, ‘‘Short paper: A look at smartphone permission models,’’ in *Proc. 1st ACM Workshop Secur. Privacy Smartphones Mobile Devices*, 2011, pp. 63–67, doi: [10.1145/2046614.2046626.](http://dx.doi.org/10.1145/2046614.2046626)

**[5].M. Kaur, D. Singh, V. Kumar, B. B. Gupta, and A. A. A. El-Latif**, ‘‘Secure and energy efficient-based e-health care framework for green Internet of Things,’’ *IEEE Trans. Green Commun. Netw.*, vol. 5, no. 3, pp. 1223–1231, Sep. 2021, doi: [10.1109/TGCN.2021.3081616.](http://dx.doi.org/10.1109/TGCN.2021.3081616)

**[6].M. Kaur and D. Singh,** ‘‘Multiobjective evolutionary optimization techniques based hyperchaotic map and their applications in image encryption,’’ *Multidimensional Syst. Signal Process.*, vol. 32, no. 1, pp. 281–301, Jan. 2021.

**[7].Android Architecture** *| Android Open Source Project*. Accessed: Nov. 22, 2020. [Online]. Available: https://source.android.com/devices/ architecture

**[8].M. Kaur and V. Kumar**, ‘‘Parallel non-dominated sorting genetic algorithm- II-based image encryption technique,’’ *Imag. Sci. J.*, vol. 66, no. 8, pp. 453–462, Nov. 2018.

**[9].M. Kaur and V. Kumar,** ‘‘Beta chaotic map based image encryption using genetic algorithm,’’ *Int. J. Bifurcation Chaos*, vol. 28, no. 11, Oct. 2018, Art. no. 1850132.

**[10].M. Backes, S. Bugiel, S. Gerling, and P. von Styp-Rekowsky**, ‘‘Android security framework: Extensible multi-layered access control on Android,’’ in *Proc. 30th Annu. Comput. Secur. Appl. Conf.*, Dec. 2014, pp. 46–55, doi: [10.1145/2664243.2664265.](http://dx.doi.org/10.1145/2664243.2664265)

**[11].S. Heuser, A. Nadkarni, W. Enck, and A.-R. Sadeghi**, ‘‘ASM: A programmable interface for extending Android security,’’ in *Proc. 23rd USENIX Secur. Symp. (USENIX Security)*, San Diego, CA, USA, 2014, pp. 1005–1019.

**[12].M. Y. Karim, H. Kagdi, and M. Di Penta,** ‘‘Mining Android apps to recommend permissions,’’ in *Proc. IEEE 23rd Int. Conf. Softw. Anal., Evol., Reeng.*, Mar. 2016, pp. 427–437, doi: [10.1109/SANER.2016.74.](http://dx.doi.org/10.1109/SANER.2016.74)

**[13]Android Platform | Android Developers**. Accessed: Nov. 24, 2020. [Online]. Available: https://developer.android.com/about

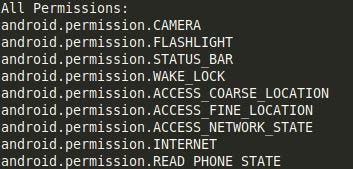
*[***14]Mobile & Tablet Android Version Market Share Worldwide** *| StatCounter Global Stats*. Accessed: Dec. 10, 2020. [Online]. Available: https://gs.statcounter.com/android-version-market-share/mobile- tablet/worldwide

**[15].B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and**

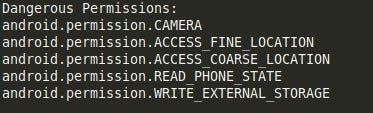
**G. Álvarez,** ‘‘PUMA: Permission usage to detect malware in Android,’’ in *International Joint Conference CISIS’12-ICEUTE’12-SOCO’12 Spe- cial Sessions*. Berlin, Germany: Springer, 2013, pp. 289–298, doi: [10.1007/978-3-642-33018-6\_30.](http://dx.doi.org/10.1007/978-3-642-33018-6_30)

**APPENDIX A-SCREENSHOTS**

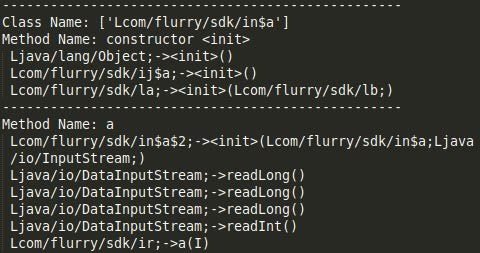
**Snapshot of a part of all permissions of brightest flashlight obtained from the proposed framework.**



**Potentially dangerous permissions of brightest flashlight identified in static analysis obtained from the proposed framework.**



**Class and corresponding method in brightest flashlight APK shown in the map obtained from the Smali code parser.**



**APPENDIX B-SOURCE CODE**

**from tkinter import \***

**import time**

**import re**

**from androguard import misc**

**import pickle**

**from PIL import Image, ImageTk**

**from tkinter.filedialog import askopenfile**

**import numpy as np**

**def pp(a):**

**global mylist**

**mylist.insert(END, a)**

**def predict(val):**

**global mylist**

**print(val)**

**a, d, dx = misc.AnalyzeAPK(val)**

**print(a.get\_permissions())**

**print(" [\*] Success")**

**perlist=a.get\_permissions()**

**pickle.dump(perlist,open("perdata.pkl","wb"))**

**cnt=500**

**for i in perlist:**

**print(i)**

**mylist.insert(END, i)**

**def browseim():**

**global cimg,shrslt,E1**

**path = askopenfile()**

**n=path.name**

**print(n)**

**E1.delete(0,"end")**

**E1.insert(0, n)**

**def userHome():**

**global root, mylist,shrslt,E1**

**root = Tk()**

**root.geometry("1200x700+0+0")**

**root.title("Home Page")**

**image = Image.open("mlwr.jpg")**

**image = image.resize((1200, 700), Image.ANTIALIAS)**

**pic = ImageTk.PhotoImage(image)**

**lbl\_reg=Label(root,image=pic,anchor=CENTER)**

**lbl\_reg.place(x=0,y=0)**

**#-----------------INFO TOP------------**

**lblinfo = Label(root, font=( 'aria' ,20, 'bold' ),text="Privacy protection Android",fg="white",bg="#000955",bd=10,anchor='w')**

**lblinfo.place(x=450,y=50)**

**lblinfo3 = Label(root, font=( 'aria' ,20 ),text="input apk path",fg="#000955",anchor='w')**

**lblinfo3.place(x=180,y=200)**

**E1 = Entry(root,width=30,font="veranda 20")**

**E1.place(x=50,y=260)**

**mylist = Listbox(root,width=60, height=20,bg="white")**

**mylist.place( x = 700, y = 200 )**

**btntrn=Button(root,padx=10,pady=2, bd=4 ,fg="white",font=('ariel' ,16,'bold'),width=10, text="Browse", bg="red",command=lambda:browseim())**

**btntrn.place(x=180, y=300)**

**btnhlp=Button(root,padx=80,pady=8, bd=6 ,fg="white",font=('ariel' ,10,'bold'),width=7, text="Submit", bg="blue",command=lambda:predict(E1.get()))**

**btnhlp.place(x=150, y=400)**

**def qexit():**

**root.destroy()**

**root.mainloop()**

**userHome()**