International Journal of Early Childhood Special Education (INT-JECSE)
DOI: 10.48047/INTJECSE/V14I6.425 ISSN: 1308-5581 Vol 14, Issue 06 2022

An Enhanced and Efficient Permission Manager for Android Automotives

V.Sivamurugan, K.R.Uthayan
[1]sivamuruganv@ssn.edu.in, [2]uthayankr@ssn.edu.in
Department of Information Technology,
Sri Siva Subramaniya Nadar College of Engineering, Chennai, India

ABSTRACT
Data privacy has always been important. As more of our data becomes digitized, and we share more information online, data privacy is taking on greater importance. A single company may possess the personal information of millions of customers' data that it needs to keep private so that customers' identities stay as safe and protected as possible, and the company's reputation remains untarnished. Data Technology (DT) Privacy Technology is a Data privacy company solely aiming to put back the data transparency and consent to the user in making every single user of the car aware of what exactly they are sharing and what should not be shared. The DT Application is based on the framework that focuses on three important aspects of data sharing: Data Transparency, Consent, and convenience. The framework is designed in a way that the DT Application has two important features in it - Data monitoring channel and Data Control channel. Apart from the above features DT Application also focuses on the system permissions. System permissions play a crucial role in the Android security architecture. They are used to restrict app operations only to resources (e.g., file system, network) that the user has agreed to share. An approach aimed at helping Android users and developers to compare the risk level of a set of Android applications. DT Application assigns risk factors to each of the applications based on the Android classification of permissions and helps users to control the permissions. This application is divided into different parts for development. For this particular work, the Permission Module, the view layer and the view model layer using Model View Controller architecture with data binding is being developed. It also enables the user to monitor and control the real time permissions (location, camera, microphone etc.) that an application is accessing to send data through the network. The proposed New Permission Manager shall give the user complete control over the applications on which permissions that they use and keep the user aware of the background permission usage situation with Live monitoring.

KEY WORDS: Android Operating System, Automotive Android, Data Control Channel, Data Transparency, Infotainment System, Permission Manager and Security Architecture.

1. INTRODUCTION

The automotive industry has come a long way with many users' centric features and facilities. One of the facilities in automotive is its smart infotainment system. The infotainment system runs on an android based Operating System and hence the user should be aware of how the data is used so the privacy is safeguarded. The existing Permission Manager does not provide the user full control and transparency. In our proposed System, the Permission Manager enables the user to fully control and get Real time alerts and notifications based on the application usage of dangerous permissions. The need for handling different permissions and its associated categories is creating ambiguities about which application uses which all permissions in android automotive OS. Definitive algorithm encompassing permission and its related services has become a problem/need in the modern android framework. The existing permission manager is not transparent enough and doesn't give the user full control over the applications. Thus, our system was built, to control the data sharing, of what to share and what not, based on the user's privacy reasons and to enable System permissions to restrict app operations only to resources that the user has agreed to share. There is no user-friendly UI to handle permissions effectively in the android framework. There is no single click control to handle multiple permissions of an application in the android framework. There is no real time indication of dangerous permission usage by the applications (Live Monitoring). The enhanced permission manager is used to control Dangerous Permissions with a more transparent User Interface and also to enable real-time monitoring of critical permissions such as (location, microphone, camera etc.) along with Voice Alerts and Voice Controls.

2. LITERATURE SURVEY

Branimir Kovacevic et. al [6] have proposed a system for integration of Android into vehicle infotainment systems. The proposed system offers infotainment functionality, while enabling users to use the device as a regular Android device. The proposal included a specification of a Java API that should be used to access in-vehicle related content from Android applications.
Abdul Moiz et. al [1] have examined 14 vulnerabilities out of which 11 were reproducible in Android Auto as these apps are basically Android mobile apps and run directly in the user's smartphone device. Whereas for Android Automotive 9 vulnerabilities were reproducible, remaining 5 were not reproducible either due to permission restrictions or due to API deprecation in Android 9.0 (Pie). They have also categorized these vulnerabilities as per their type and provide their severity levels. For some of the vulnerabilities they have provided a compliant solution which can be useful to mitigate some vulnerabilities while others, like accessing precise location information and deriving speed from it, varies from app-to-app usage of that information.
Abdul Moiz et. al [2] have investigated the security concerns of in-vehicle apps, specifically, those related to inter component communication (ICC) among these apps. ICC allows apps to share information via inter or intra apps components through a messaging object called intent. In case of insecure communication, Intent can be hijacked or spoofed by malicious apps and user’s sensitive information can be leaked to a hacker's database. They have investigated the attack surface and vulnerabilities in these apps and provided a static analysis approach and a tool to find data leakage vulnerabilities. The approach also provides hints to mitigate these leaks. They have evaluated their approach by analysing a set of Android Auto apps downloaded from Google Play store, and we reported the validated results on vulnerabilities identified on those apps. Our proposed approach integrates different classes of automotive applications and enables secure inter-system communication on top of container-based technology. Srdjan Usorac et. al [16] have proposed an approach that integrates different classes of automotive applications and enables secure inter-system communication on top of container-based technology. Their future work will expand the implementation of our current solution with additional Android features, to run more demanding Android applications in a secure Linux environment. Edwin Franco Myloth Josephlal et. al [8] have focused on identifying the vulnerabilities of the automotive infotainment system with respect to its WIFI capabilities by conducting structured vulnerability tests on the WIFI capabilities of an automotive infotainment system. To do this, they have analysed the WIFI attack surface and constructed test environments and used appropriate tools such as (Nmap (open port scan), Nessus (vulnerability scan), Metasploit) to generate a penetration testing plan to search for vulnerabilities. The vulnerability findings are well documented in their work. Marco De Vincenzi et. al[13] have described the app structure, its operations, and the E-Corridor architecture, where vehicle data are analysed. They have also provided an insight into the possible privacy concerns caused by the usage of personal and vehicle data. The result is an Android app that has been tested in the project infrastructure with a real vehicle, and that can be used as a general schema to create other rewarding apps, also for autonomous vehicles, using driving data.

Bogdan Groza et. al [5] have advocated a role-based access control policy mixed with attributes that facilitates access to various functionalities of vehicular on-board units from smartphones. They have also used a rights-based access control policy for in-vehicle functionalities like the case of a file allocation table of a contemporary OS, in which read, write or execute operations can be performed over various vehicle functions. Further, to assure the appropriate security, they have also developed a protocol suite using identity-based cryptography and they relied on group signatures which preserve the anonymity of group members thus ensuring privacy and traceability. David Herges et. al[7] have proposed Ginger -- an access control framework for telematics applications. Ginger is context aware, provides enhanced privacy protection, and realizes advanced access control paradigms. They have also demonstrated Ginger's feasibility with an Android-based implementation in a functional evaluation consisting of two representative use cases and in a comparative analysis with related approaches. Maryam Nijafi et. al[14] have proposed a graph-based model to determine abusive applications by automatically analysing the requested permissions. Their work aims to build a confidence indicator to choose the applications with more respect for privacy. This model would also inform the user about the possibility of data leakage risks by assigning a privacy score. Pese et.al [15] have presented an Android Automotive system architecture and provides guidelines for conducting a high-level security analysis. They have also described what countermeasures have already been taken by Google to prevent potential attacks and discuss what still needs to be done in order to offer a secure and privacy-preserving Android experience for next-generation IVI platforms. Macario et. al [12] have presented a proof-of-concept architecture developed in cooperation between Magneti Marelli and Politecnico di Torino, whose main contribution is an automotive-oriented extension of Google Android that provides features for combining extensibility and safety requirements.

Li, Li, Bissyande et. al [11] have empirically investigated 17 important releases of the Android framework source code base, and they found that inaccessible APIs are commonly implemented in the Android framework, which are further neither forward nor backward compatible. Moreover, a small set of inaccessible APIs can eventually become publicly accessible, while most of them are removed during the evolution, resulting in risks for such apps that have leveraged inaccessible APIs. Finally, they showed that inaccessible APIs are indeed accessed by third-party apps, and the official Google Play store has tolerated the proliferation of apps leveraging inaccessible API methods. John Businge et. al[9] have analysedthe survival of 467 Eclipse third-party plug-ins altogether having 1,447 versions. They have classified these plug-ins into two categories: those that depend on only stable and supported Eclipse APIs and those that depend on at least one of the potentially unstable, discouraged, and unsupported Eclipse non-APIs. Comparing the two categories of plug-ins, they observed that the plug-ins depending solely on APIs had a very high source compatibility success rate compared to those that depend on at least one of the non-APIs. They have also observed that recently released plug-ins that depend on non-APIs also have a very high forward source compatibility success rate. This high source compatibility success rate is due to the dependency structure of these plug-ins: recently released plug-ins that depend on non-APIs predominantly depend on old Eclipse nonAPIs rather than on newly introduced ones. Finally, they showed that the majority of plug-ins hosted on SourceForge do not evolve beyond the first year of release.

Kathy Wain Yee Au et. al [10] have performed an analysis of the permission system of the Android smartphone OS in an attempt to begin answering some of these questions. Because the documentation of Android's permission system is incomplete and because they wanted to be able to analyse several versions of Android, they developed PScout, a tool that extracts the permission specification from the Android OS source code using static analysis. PScout overcomes several challenges, such as scalability due to Android's 3.4-million-line code base, accounting for permission enforcement across processes due to Android's use of IPC, and abstracting Android's diverse permission checking mechanisms into a single primitive for analysis. Alexandre Bartel et. al [4] have shown that naive static analysis fails miserably when applied with off-the-shelf components on the Android framework. They have then presented an advanced class-hierarchy and field-sensitive set of analyses to extract this mapping. Those static analyses can analyse the Android framework. They have used novel domain specific optimizations dedicated to Android. Adrienne Porter Felt et. al [3] have studied Android applications to determine whether Android developers follow least privilege with their permission requests. They have built Stowaway, a tool that detects over privilege in compiled Android applications. Stowaway determines the set of API calls that an application uses and then maps those API calls to permissions. They have used automated testing tools on the Android API in order to build the permission map that is necessary for detecting over privilege. They have also applied Stowaway to a set of 940 applications and find that about one-third are overprivileged. We investigate the causes of over privilege and find evidence that developers are trying to follow least privilege but sometimes fail due to insufficient API documentation.

The Existing Permission manager is derived from Google permission manager. It is not possible to control more than one application at the same time using the existing model. There is no real-time monitoring when an application uses a permission and gets data. The system applications such as google play store, google maps, google calendar etc. can't be controlled and by default they are given all the permissions that are required to get complete control over the device. There is no indication whether the permissions are used in the background or foreground. The proposed system is designed to overcome all these limitations.

3. PROPOSED SOFTWARE ARCHITECTURE METHODOLOGY

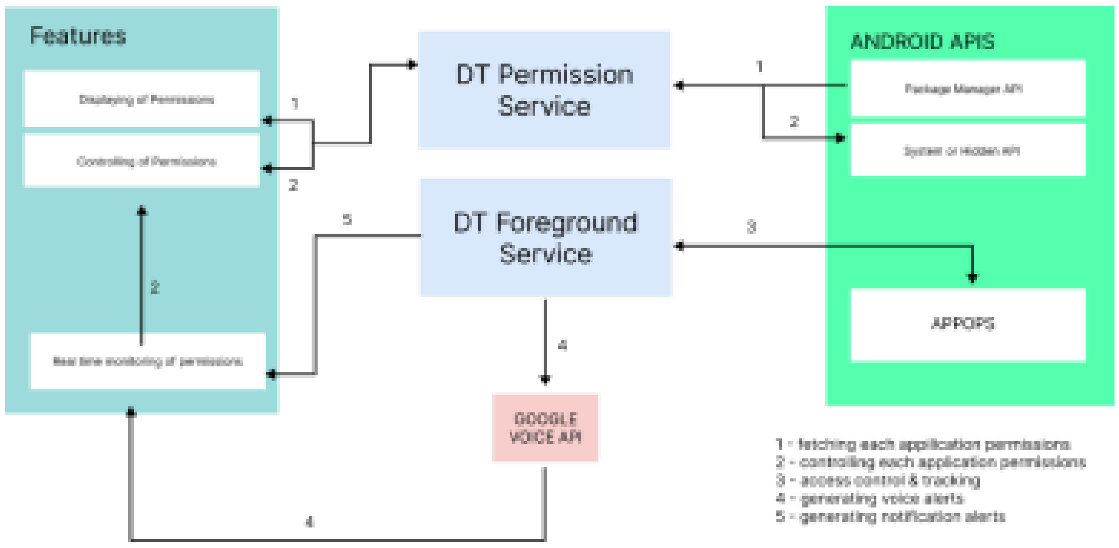

Features
Displaying of Permissions
Controlling of Permissions
Real time monitoring of permissions
DT Permission Service
DT Foreground Service
GOOGLE VOICE API
ANDROID APIS
Package Manager API
System or Hidden API
APPOPS
1 - fetching each application permissions
2 - controlling each application permissions
3 - access control & tracking
4 - generating voice alerts
5 - generating notification alerts

Figure 3.1 Proposed Software Architecture

The software architecture in Figure 3.1 shows the outline of different steps involved in the working of data technology (DT) Application. The different features provided by them are Displaying of different Permissions, Controlling the Permissions and Real time monitoring of permissions. These are achieved using the two major blocks namely DT Permission Service and DT Foreground Service shown in the above architecture. To display the permissions of a particular application, the DT PermissionService (DTPS) is called. DTPS fetches data from Package Manager API which is part of android APIs. To control the permissions, DTPS uses Permission Manager APIs (System or Hidden APIs). The System APIs are only visible and accessible to the system applications such as settings for security purposes. Third Party applications are not allowed to access those APIs which will result in exploitation of the whole system.
Real time monitoring is a novel feature that we have incorporated in a DT application. For monitoring the permission usage of an application, the DT Foreground Service is used (DTFS). DTFS runs always in the background, monitoring the permission activities of the applications through App-ops in android automotive. Each application in the android architecture has to notify App-ops if they want to use a permission. The App-ops are used for access control and tracking. DTFS listens to the App ops for permissions usage and notify the user with alerts in the form of notification and voice alerts.
Notifications are generated and the respective permission can be revoked from the application if needed. For that particular purpose again DTPS is called and desired action is executed. Google Voice APIs are called for voice generation and controlling of the permissions.

3.2 DT SERVICES

3.2.1 DT PERMISSIONS SERVICE

DT Permission Service is a service whose prior job is to fetch and control the permission of all the applications installed or specific applications in the android automotive. The DT Permission Service aggregates and segregates the permissions such as System Permissions and Car Permissions. This service is also used to control multiple applications at once or multiple permissions at once. It also does control the permissions by using System APIs.

3.2.2 DT FOREGROUND SERVICE

DT Foreground Service is a service whose prior job is to monitor the permissions of the applications that are installed in android automotive. The DTFS listens to the App-ops, which has the data of which application requested which permission at a particular instance of time. This service generates a notification alert with the respective application permission usage. To control the permission, it calls DT Permission Service. The alert is automatically discarded when the permission is no longer used and created if it is used again.

3.2.3 GOOGLE VOICE APIS

3.2.3.1 VOICE ALERTS

Google Speech Services enable text-to-speech ability which is used here to alert the user with a Voice Alert.

3.2.3.2 VOICE CONTROL

Google Speech to Text APIs with deep links and actions and controlling of permissions over voice is enabled.

3.2.4 PACKAGE MANAGER APIS

Package Manager API is used for retrieving various kinds of information related to the application packages that are currently installed on the device. Package Manager includes the application's name, description, uid, permissions related information etc.

3.2.5 PERMISSION MANAGER API

Permissions Manager API are hidden or System APIs. They are hidden from the third-party application developers. A third-party application can't use hidden APIs. This Permission Manager API contains the necessary APIs to grant and revoke permission from a specific application. These are accessed only by the system applications or signed applications. The signed applications obtain the status of system application once they are signed with the key that is specific to the AOSP build.

3.2.6 APPS-OPS

App-ops are used for two purposes: Access control and tracking. App-ops cover a wide variety of functionality from helping with runtime permissions, access control and tracking to battery consumption tracking. App-ops can either be controlled for each uid or for each package. Which one is used depends on the API provider maintaining this app-op. For any security or privacy related app-op the provider needs to control the app-op for per uid as all security and privacy is based on uid in Android. App-ops permissions are platform defined permissions that can be overridden. The security check for app-op permissions should by default check the permission grant state. If the app-op state is set to MODE_ALLOWED or MODE_IGNORED, the app-op state should be checked instead of the permission grant state. Figure 3.2 shows the Appops integration.

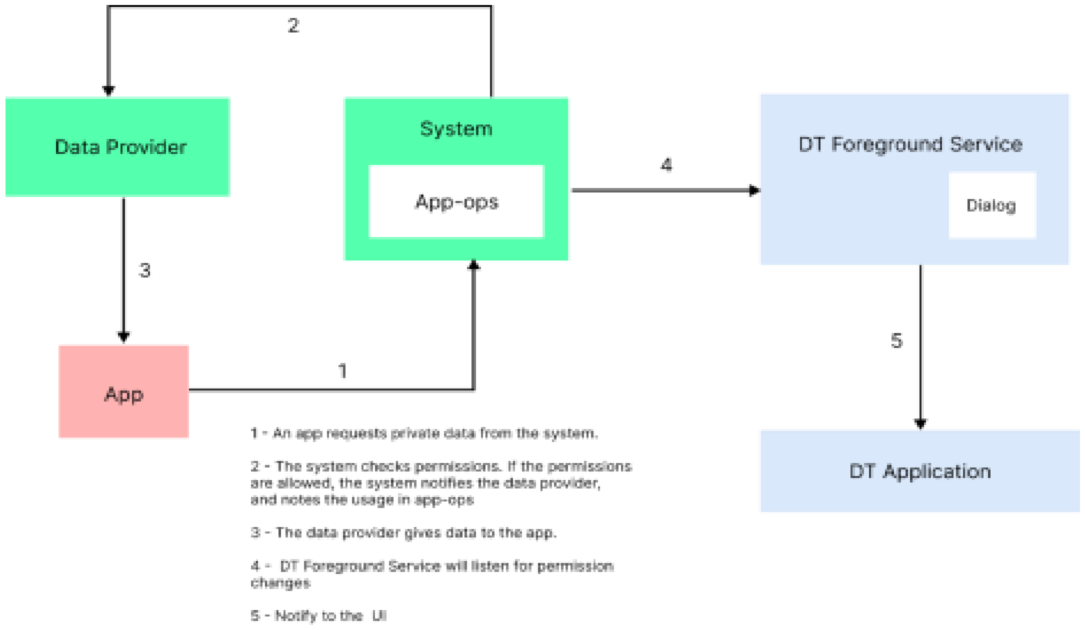

2
Data Provider
System
App-ops
4
DT Foreground Service
Dialog
3
5
App
1
DT Application
1 - An app requests private data from the system.
2 - The system checks permissions. If the permissions are allowed, the system notifies the data provider, and notes the usage in app-ops
3 - The data provider gives data to the app.
4 - DT Foreground Service will listen for permission changes
5 - Notify to the UI

Figure 2 AppOps Integrations

This functionality allows access by default to apps fulfilling the requirements for a certain permission level. Still the behaviour can be overridden when needed. App-ops track many important events, including all accesses to runtime permission protected APIs. This is done by tracking when an app-op was noted or started. The tracked data can only be read by system components.
Only noteOp(String, int, String)/startOp(String, int, String) are tracked; unsafeCheckOp(String, int, String) is not tracked. Hence it is important to eventually call noteOp(String, int, String) or startOp(String, int, String) when providing access to protected operations or data. Some apps are forwarding access to other apps. E.g., an app might get the location from the system's location provider and then send the location further to a 3rd app. In this case the app passing on the data needs to call noteProxyOp(String, String) to signal the access proxying. This might also make sense inside of a single app if the access is forwarded between two parts tagged with different attribution tags. An app can register an OnOpNotedCallback to get informed about what accesses the system is tracking for it. As each runtime permission has an associated app-op this API is particularly useful for an app that wants to find unexpected private data accesses.

3.3 Front-End Architecture

MVVM (Model, View, View Model) Architecture shown in Figure 3 helps to achieve modularity and so that the View, Business Logic (View Model) and Data Source are separated. Activity or Fragment is the View and binded with the View Model and enables continuous streamline transfer of data from the data source. Repository is the Interface layer between the View Model and the Database. The View Model doesn’t have any knowledge of where the data is coming from. The repository relates to the database and initializes the database using Data Access Object (DAO). DAO is a database interface that helps the repository to communicate with the SQL database. Data Source may be Database or other android APIs such as Package Manager etc.

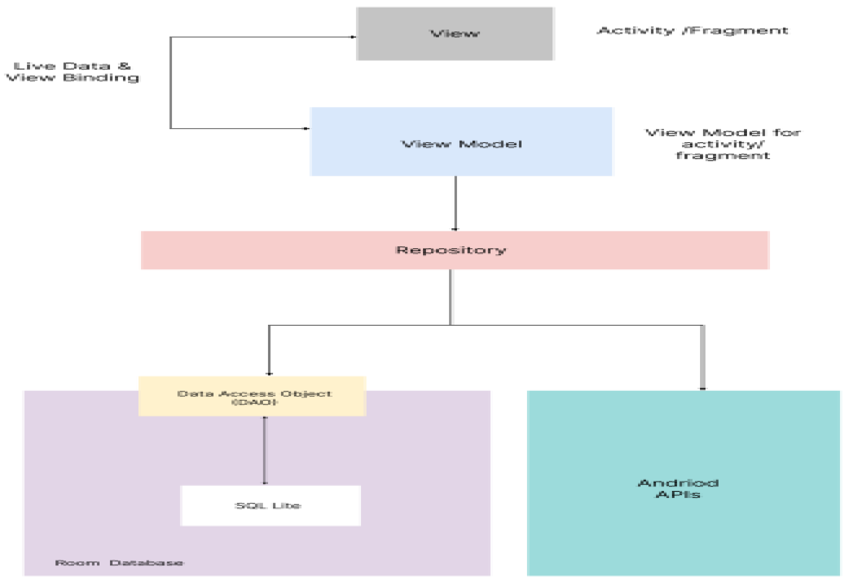
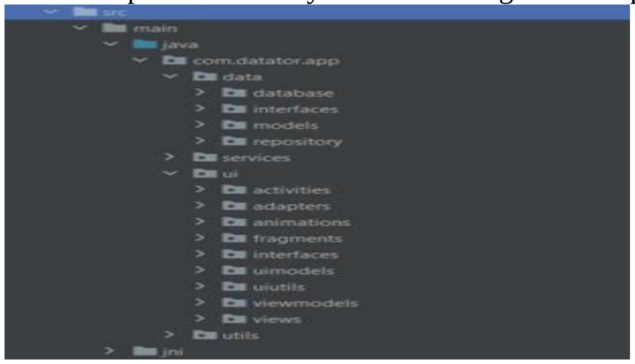


Figure 3. MVVM Architecture

The View can be completely replaced without changing any of the Business Logic, data source and vice versa. MVVM provides complete modularity as shown in Figure 4 and provides data abstraction.



```
src
  main
    java
      com.datator.app
        data
          database
          interfaces
          models
          repository
        services
        ui
          activities
          adapters
          animations
          fragments
          interfaces
          uimodels
          uiutils
          viewmodels
          views
        utils
    jni
```

Figure 4 MVVM modularity

The project structure contains three major modules: data, services, ui, utils. The module data contains four modules such as data access objects, interfaces, models, repositories. Services contain Broadcast Receiver, DT Permission Service and DT Foreground Services. UI module consists of several modules such as activities, adapters, animations, fragments, interfaces, ui models, uiuitls, view models, views.

4. RESULTS AND SNAPSHOTS FOR IMPLEMENTATION

Figure 5. is used to show the user interface designed using an Android application. This user interface shows the different applications installed in the Android Automotive Device. This user interface is used to access the different devices and to enable the different permissions.

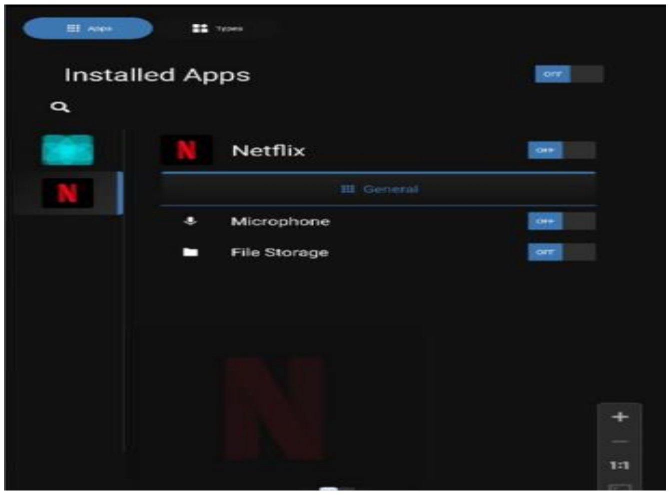
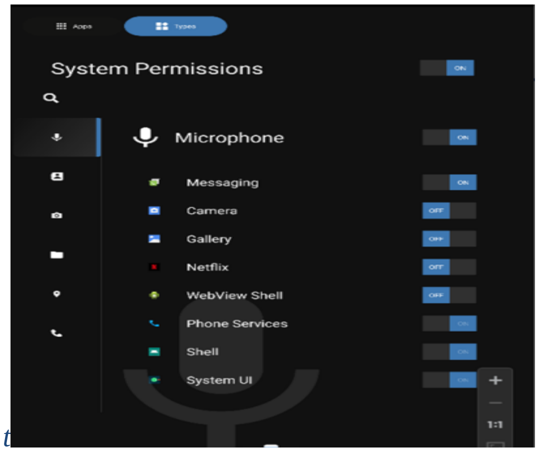

Apps
Types
Installed Apps OFF
Netflix OFF
General
Microphone OFF
File Storage OFF

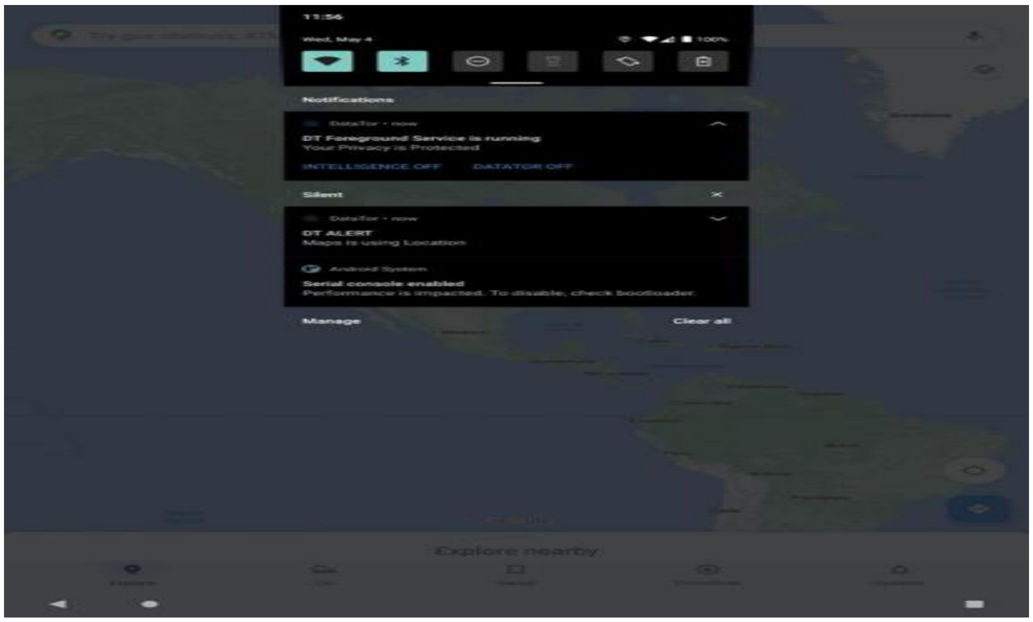
Figure 5. UI showing different installed application.


Apps
Types
System Permissions ON
Microphone ON
Messaging ON
Camera OFF
Gallery OFF
Netflix OFF
WebView Shell OFF
Phone Services ON
Shell ON
System UI ON

t

Figure 6. UI with different permissions

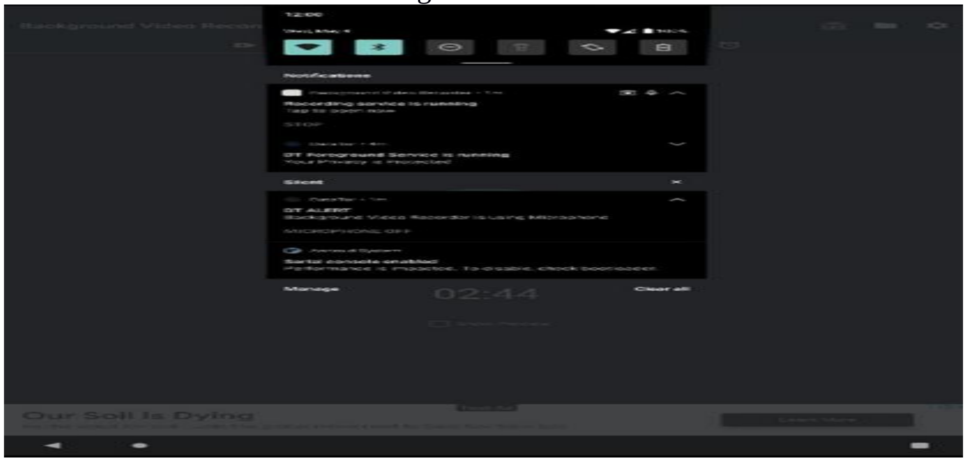
Figure 6 shows the UI with different permissions for the device namely microphone. The system permission defines a set of access rights to control the various operations on an application element or device. The access rights are ACCESS, MODIFY, DELETE, CREATE and PERMISSION. By default, run time permissions are not granted. The app needs to call Activity. requestPermissions during runtime to ask the user for the permission. The user might then grant or deny and once the decision is made the activity is called via Activity.



11:56
Wed, May 4
Notifications
DataTor • now
DT Foreground Service is running
Your Privacy is Protected
INTELLIGENCE OFF DATATOR OFF
Silent
DataTor • now
DT ALERT
Maps is using Location
Android System
Serial console enabled
Performance is impacted. To disable, check bootloader.
Manage Clear all
Explore nearby

Figure 7.UI showing Notification and Alert

Figure 7 shows the user interface showing Notification and alert messages. They are displayed when DT foreground service is running. The DTFS listens to the App-ops. This App-ops has the data about the different applications requested along with permission at a particular instance of time. This service generates a notification alert with the respective application permission usage. To control the permission, it invokes the DT Permission Service. The alert is automatically discarded when the permission is no longer used or needed and created if it is used again.



12:00
Notifications
Recording service is running
STOP
DT Foreground Service is running
Your Privacy is Protected
Silent
DT ALERT
Background Video Recorder is using Microphone
MICROPHONE OFF
Android System
Serial console enabled
Performance is impacted. To disable, check bootloader.
Manage Clear all
02:44
Our Soil Is Dying

Figure 8. showing UI activity

Figure 8 shows the UI design based on either activity or fragment. This is used to get the permission from the user and to save the permission. Thus, permission for the different devices associated with the application is either enabled or disabled based on the permission granted by the user.

5. CONCLUSION AND FUTURE WORK

Smart infotainment systems are the focus of this work. The existing Permission Manager does not provide the user full control and transparency. Therefore, the user should be aware of how the data is used so the privacy is safeguarded. In our implemented System, the Permission Manager enables the user to fully control and get Real time alerts and notifications based on the application usage of dangerous permissions. This work can be further enhanced by adding Android system intelligence to the system. This will enable us to use the system permission to provide smart predictions.

REFERENCES:

1. Abdul Moiz and Manar H. Alalfi, “An Approach for the Identification of Information Leakage in Automotive Infotainment systems”, IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM), Vol. 2020, pp. 110-114, 2020.
2. Abdul Moiz and Manar H. Alalfi, “ A Survey of Security Vulnerabilities in Android Automotive Apps”,IEEE/ACM 3rd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS), Vol.2022, pp. 17-24, 2022.
3. Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song and David Wagner, "Android permissions demystified", Proceedings of the 18th ACM conference on Computer And communications security, pp. 627-638, 2011.
4. Alexandre Bartel, Jacques Klein, Martin Monperrus and Yves Le Traon, "Static analysis for extracting permission checks of a large scale framework: The challenges and solutions for analyzing android", Software Engineering IEEE Transactions on, vol. 40, no. 6, pp. 617-632, 2014.
5. Bogdan Groza, Tudor Andreica, Adriana Berdich, Pal-Stefan Murvay and Eugen Horatiu Gurban, “PRESTvO: PRivacy Enabled Smartphone Based Access to Vehicle On-Board Units”, IEEE Access, Vol.8, pp. 119105-119122, 2022.
6. Branimir Kovacevic, Marko Kovacevic, Tomislav Maruna and Davor Rapic, “ Android4Auto: A proposal for integration of Android in vehicle infotainment systems”, IEEE International Conference on Consumer Electronics (ICCE), Vol.2016, pp. 99-100, 2016.
7. David Herges, Naim Asaj, Bastian Könings, Florian Schaub and Michael Weber, “Ginger: An Access Control Framework for Telematics Applications”, IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, Vpl.2012, pp. 474-481, 2012.
8. Edwin Franco Myloth Josephlal and Sridhar Adepu, “Vulnerability Analysis of an Automotive Infotainment System's WIFI Capability, IEEE 19th International Symposium on High Assurance Systems Engineering (HASE), Vol.2019, pp. 241-246, 2019.
9. John Businge, Alexander Serebrenik and Mark van den Brand, “Survival of eclipse third-party plug-ins", Software Maintenance (ICSM) 2012 28th IEEE International Conference on, pp. 368-377, 2012.
10. Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang and David Lie, “Pscout: analyzing the android permission specification", Proceedings of the 2012 ACM conference on Computer and communications security CCS '12, pp. 217-228, 2012.
11. Li, Li, Bissyande, Tegawende F, Traon , Yves Le and Klein, Jacques., “Accessing inaccessible Android apis: An empirical study,” 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), doi: 10.1109/ICSME.2016.35.
12. Macario, M. Torchiano and M. Violante, "An in-vehicle infotainment software architecture based on google android," 2009 IEEE International Symposium on Industrial Embedded Systems, 2009, pp. 257-260, doi: 10.1109/SIES.2009.5196223.
13. Marco De Vincenzi, “DRIVES: Android App for Automotive Customized Services”, 2022 IEEE/ACS 19th International Conference on Computer Systems and Applications (AICCSA), Vol.2022, pp. 1-8, 2022.
14. Maryam Nijafi, Marc Lemercier & Lyes Khoukhi, “Data leakage prevention model for vehicular networks”, 18th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Vol.2022, pp. 124-129, 2022.
15. Pese, M., Shin, K., Bruner, J., and Chu, A., “Security Analysis of Android Automotive,” SAE Technical Paper 2020-01-1295, 2020, doi:10.4271/2020-01-1295.

16. Srdjan Usorac and Bogdan Pavkovic, “Linux container solution for running Android applications on an automotive platform”**,** Zooming Innovation in Consumer Technologies Conference (ZINC), Vol.2021, pp. 209-213, 2021.