JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Improved Permission Access Management Using Logistic Regression, Machine Learning Approach on Android Apps

A critical look at the implicit and explicit intent mechanism for communication between application components and its associated security challenges on Android Platform.

Frank Laud Boateng
Lecturer
School of Physical Sciences,
Kessben University College, Kumasi, Ghana

Abstract: The Android Operating System permission framework prohibits inappropriate applications from manipulating handheld interactive content and triggering critical APIs. Nevertheless, there are still several shortcomings in the existing permission administration that contribute to the authorization process being unable to adequately protect the personal data of users. In this report, a new Android permission control strategy built on machine learning is presented to mitigate the challenges of the current permission system. To realize fluid management, the suggested system retains a variable permission governance database that registers the status of the privileges on each application. Mostly the privilege given in the database context can be used in this method. In the entire process, the system initially classifies the query by machine learning, then collects the appropriate permission details from the databases and sends an unsafe access alert to users. Eventually, the framework changes the fluid control database per the user's judgments. Using this method, users can avoid the malicious actions of application obtaining personal data and triggering critical APIs on time. The system enhances the efficiency of permission administration and strengthens the protection and stability of interactive data on Android devices. Improved Permission Access Management Mode (iPAM) is presented as proof of concept, and alternative tool to manage and administer permission as request by applications both malign and malicious applications that seeks to exploit components of other vulnerable applications to cause harm or breach of users' privacy. The proposed model iPAM integrates machine learning and privilege administration to strengthen the default privilege framework as set out in the Android Operating System Permission Framework. Translating this model into the real-world environment and on a physical Android device is important in the future.

Keywords **– iPAM, AOS, Android, Permission, Authorization, Platform, Vulnerability, APIs.**

I. INTRODUCTION

Smart devices include smartphones, laptops and Personal Digital Assistant (PDA) is the current generation of consumer electronic devices, although they have been around since the beginning of the 1990s (Backes et al., 2013; Tuncay et al., 2018; Zhang et al., 2016). Smartphones integrating telephony and telecommunications computing have emerged in recent decades as a popular trend in electronic products across the world today [4]. In the early years of the '90s, the mobile phone was just normal phones that could only be used to place calls and receive the same and nothing else, and computers were just immobile devices. Therefore, it was not

possible at that time to move around computers as it is today without some sort of complicated disassembly and some fragile parts could be destroyed in the process [5].

Mobile devices are a family of devices providing more programming skills on an ordinary mobile phone; running a consummate and autonomous Operating System (OS) offering a homogeneous interface and a stage for third-party software developers [6], [7].

Today's smartphone incorporates convenience, accessibility, and entertainment due to mobile features such as 2G, 3G, 4G, LongTerm Evolution (LTE) connectivity, touch screen, accelerometer, Global Positioning System (GPS) and more than 1.7 GHz processing speed and power. [8]. screen, according to (Khokhlov et al., 2018; Ren et al., 2015). Distinct innovations in the OS that are commonly used allow distinct markets for applications for these various platforms; the advantage of alternatives for the average consumer as shown in Figure 1

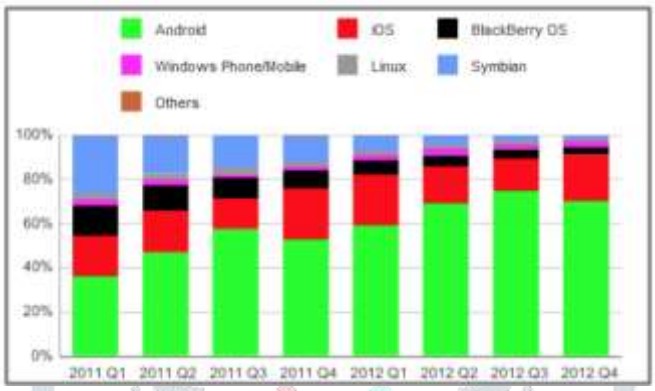


Figure 1: Showing percentage of used OS [11]

[12] also noted that the two biggest smartphone platform OS in the contest today is Apple's iOS and Google's Android as presented in Figure 1.1 with 75% of the consumers using Android-based smartphones which makes it the big cheese rival. Nevertheless, Android has had a challenge with up-to-date old devices with late OS versions as they leave that to Original Equipment Manufacturer (OEM) (Boksasp et al., 2012). Many Android-based devices today are still running older versions of the OS platform according to [14].

This is a challenge because old found liabilities and exploitations are as threatening as when they were initially found or advanced [15]. With no patches for the liabilities either from OEM or Google as the main maintainer of the OS, it is practicable to use these aged liabilities for late exploitation as if the liabilities where zero-day old [16]. With time these exploits and liabilities just keep appending and piling up and endlessly get patched, that begets an outsize measure of feasible attack vectors, and staying protected becomes almost impossible for the devices that are still running older versions of the OS as shown in Figure 2.

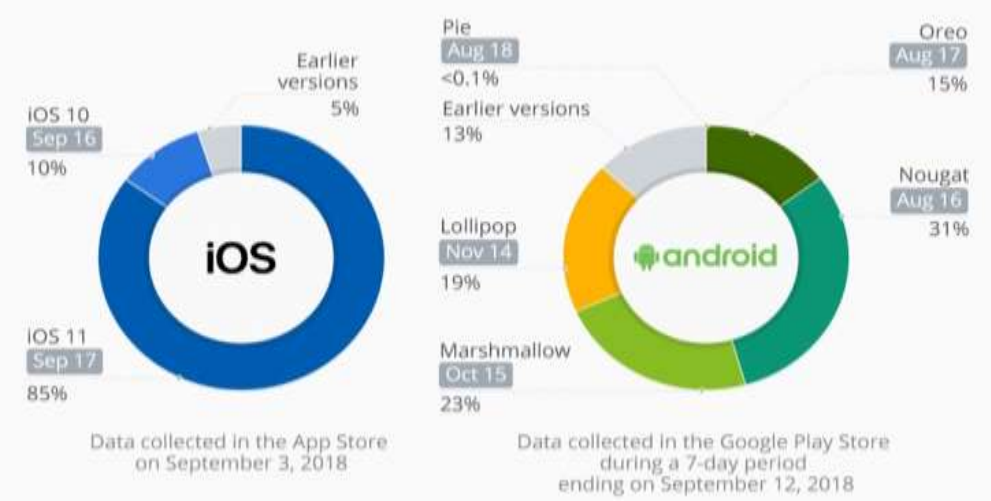


Figure 2: Shows Update Frequency of iOS and Android (Cauchi et al., 2018).

II. PROBLEM STATEMENT

The Android permission system restricts application that has been installed on a device and prohibits it from accessing the critical system resources and user personal information by invoking some sensitive API of which it does not have the requisite permission. There are still plenty of shortcomings in the new permission administration, though, which means that the default process can not sufficiently protect the private information of users [18].

In this study, Android's alternative permission management model is implemented, based on machine learning, to resolve the current permission management drawbacks. To accomplish the fluid permission control, the proposed model maintains a database that tracks the permission status for each application. Only the permission given in the database can be used in the requested application.

III. SIGNIFICANCE OF THE STUDY

Android marketplace maintains a wide range of third-party applications and users can install applications with varying degrees of confidence. End-users and developers install applications from both known and unknown developers of software along with applications capable of handling confidential user data such as medical records [12], [19].

Given all their default isolation policy, applications can communicate discretionarily through a message-passing mechanism called Intent Communication, through this medium can become an attack vector. When third party software developers reveal methods, classes, and functionalities actively or passively, the application may be tricked into performing an unwarranted task or action [20].

The proposed model aim at acting as Man-in-the-Middle in this communication channel, with the sole purpose to test whether the component has the privilege as in permissions to either request or perform an action or execute a task [4], [21], [22]). This will ensure that applications and their components have the required permissions and privileges to carry out an operation or collect remotely relevant data and no other application or components. To this extent, users may be secure and be protected as all these Intent communications are virtually not visible to the user [14].

IV. SCOPE OF THE PROJECT

This study focuses on the Android OS since it has about 75% of the market space of mobile telephone and smart device operating system platform. The theory about threats will as a starter focuses on Android 2.3.3 and 4.0.2 Jelly Bean up to 6.0.1 (Marshmallow) Application Programming Interface (API 23) since about 54% cannot get the latest Android OS 8.0 or Android P (Oreo, API 27, Android P API 28) [23].

V. METHODOLOGY

The study aims to develop an improved model for managing access to permissions. The permission access model of the Android program package is proposed and applied after research on the Android permission framework as shown in Figure 3. The entire model consists of the module for application classification and the module for permission management. The application classification module will initially retrieve permissions and sensitive API data from a decompiled APK file. These two excerpts are considered a key attribute for the classification of applications. The permission management module measures the classifier to evaluate the category to which the application belongs and then provides the user with the permission warning according to the unique access whitelist category. In the end, the module prompts the user to agree to the granting or otherwise of the requested permissions.

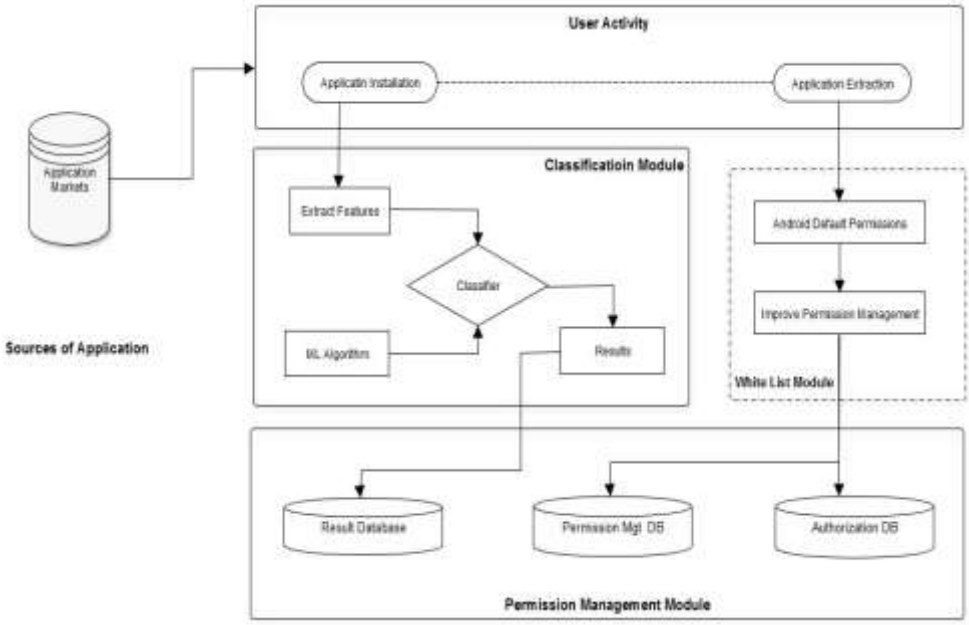


Figure 3: iPAM Model Architecture.

- **Classification Module.**

Classifier's accurate and efficient work provides the entire model with an effective execution guarantee. Classification methodology is commonly used to solve problems in a wide range of areas, even in medical settings. Many machine learning algorithms efficiently represent sequential observations and tend to cluster between separate elements. This section introduces the creation of the application classifier module to a large extent and explains every step in the process [24].

- **Application Classifier Training.**

Training and testing of the module on MacBook Pro help to obtain the application classifier as the development machine, the most important and complex process of the permission administration module, is the use of a substantial number of APK programs. The machine learning classification approach involves the selection of APK data set, data pre-processing, data modelling, extraction and feature value processing, training, and testing, as shown in Figure 4. Such steps are outlined as follows.

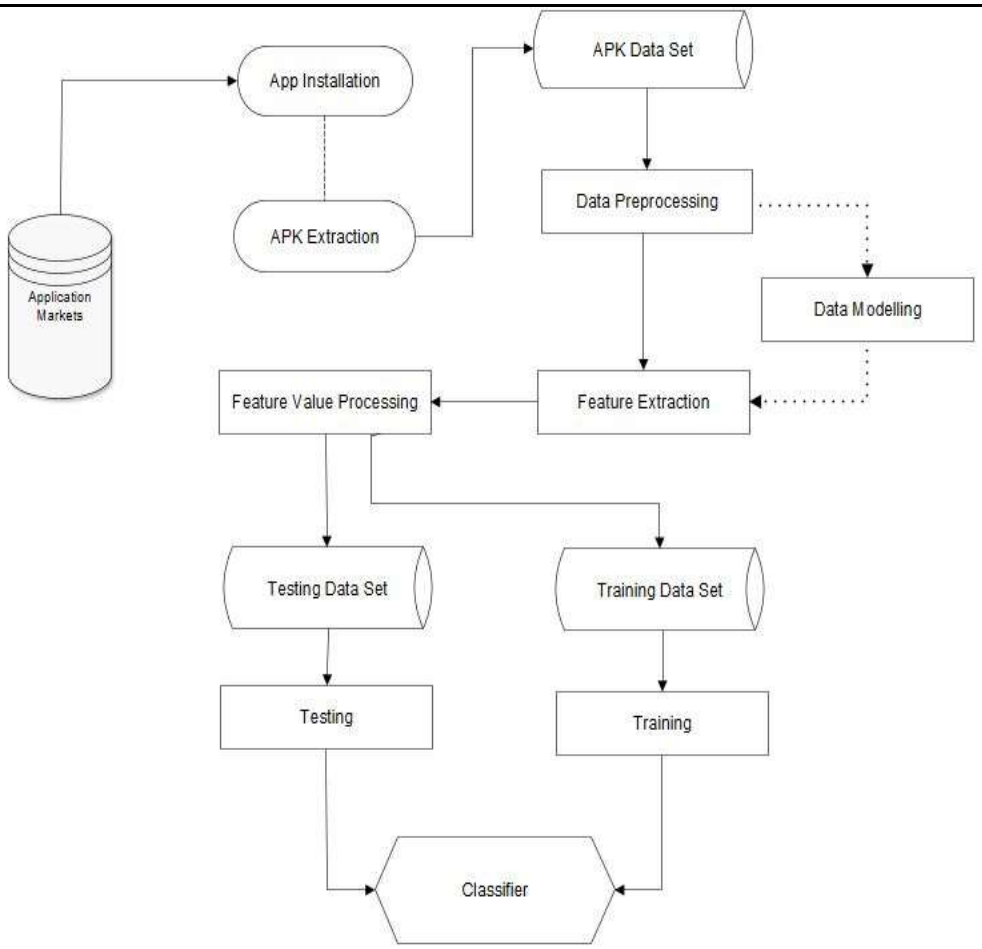


Figure 4: Flowchart of iPAM Application Classification Training.

- **Data Pre-processing.**

Google Play Store, which is the most common application platform for selecting APK data sets was used. The data sets are processed in advance and placed into separate categories. Ultimately, some 400 APK data sets were gathered in total and all these data sets are put in separate categories. The application classification comprises 10 segments including Sports, Health and Fitness, Education, Books, Media, Music, Social Networking, Family, Travel, and Tools

- **Data Modelling.**

Since similar types of Android apps have similar features, it's easy to figure out that they're using similar APIs and permissions. This work thus subscribes to the inclusion of permissions and sensitive APIs as the feature value in data modeling implementation. This study implements the space vector model that transforms the APK file into a multidimensional vector relative to the value of the function elements, and each element corresponds to a single vector dimension. The weights of the function elements are calculated using a binary approach [25]. Here, the data sets collected are presumed to be given as

$$(P_1, A_1, C_1), \ldots (P_i, A_i, C_i), \ldots (P_n, A_n, C_n).$$

For example, where Pi represents the privilege use scenario of No. i APP, P1= (0, 1, 0, 1, 1) means APP1 encapsulates second, fourth and fifth privileges, Ai reflects the use of sensitive API, which incorporates both the first and third sensitive APIs. For instance, C_i {10 categories of applications}, C_1= Education which defines APP1 as per the category of education. Relating to the earlier method, the details of the APK can be adequately represented in vectors.

- **Processing of Feature Values.**

The selection and assessment of feature values is by far the most time-consuming and difficult task for the classification of applications. Through decompiling the APK file, the privileged information can be retrieved from the AndroidManifest.xml file.

First, the use of Apktool to decompile sensitive API data in the APK file, evaluate the Smali file to access the API used in the APK, then use the API library [26].

- **Training and Testing Data Set.**

Within the machine learning classification algorithm, the complete data set will be split into a training set and test set. The training set is used to train the model to construct the classifier for a particular classification algorithm. The test set is used to verify the detection ability gained by the classifier. This study uses an open-source machine-learning algorithm to train and test the data set, such as the logistic regression algorithm.

- **Classification Module Implementation**

To identify the application when it is installed, the application classification module accesses the generated classifier and then the outcome is stored in the classification result database. The program classification module outlines procedures as shown in Figure 5.

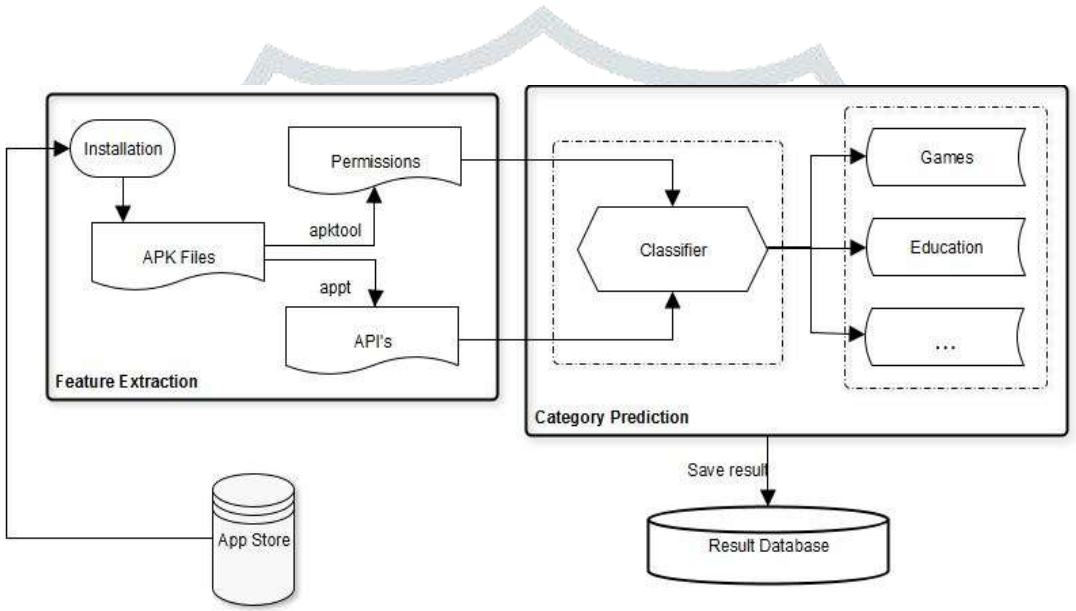


Figure 5: Flowchart of iPAM Classification Module

- **Feature Values Extraction**

The data collected from the APK applications require permissions and sensitive APIs to be used in the process of classification. Consequently, the collected feature data must be converted to the feature value that the classifier can acknowledge.

- **Classification Prediction**

The first stage of the predicting classification process is to attach the permission and important API data in the classifier. The classifier validates the classification result and then specifies which category that application belongs to and subsequently generates it.

- **Saving Classification Result**

This phase is primarily intended to store the outputs of the classification process in the database of classification results. The database contains one classification result table for all applications configured within the application classifier's expected context, which is then used to store the classification results. This primarily involves the application's UID, the name of both the application, and the group or category which the application belongs to.

VI. iPAM MODEL

Improving the module permission access management strengthens the OS's existing permission system which makes it more precisely balanced. The module dynamically grants privileges when the application is running and declines a permission, and even notifies users of the details of the permissions present. This phase involves database design and construction, user interface design, and improved permission management implementation.

- **Database Design**

The enhanced access privilege administration model focuses primarily on three-tier databases: the classification results database, the permission management database, and the permission authorization database as shown in Figure 6.

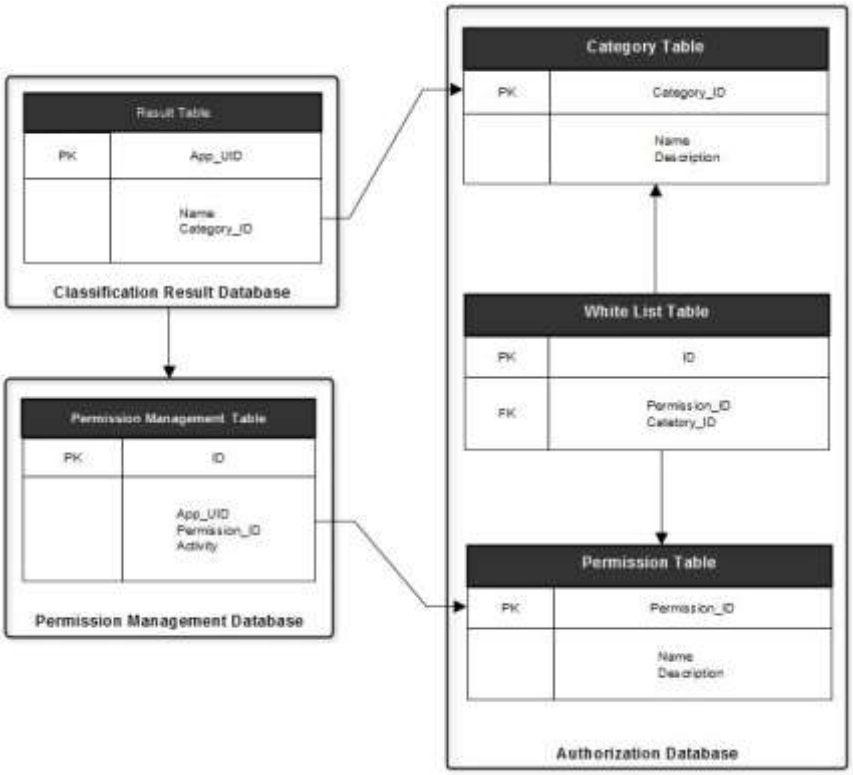


Figure 6: iPAM Database Module.

- **Authorization Database**

The database for permission authorization includes three different tables. The table for application categories stores the category of application data. Individual categories have similar functions for implementation and the same intent. All permissions data are listed in the Android OS permission table. The permission whitelist table stores a maximum of 5 permissions for each application category, considered to be the whitelist permissions for the application category. If the permission request is not included in the whitelist table, the permission management module will notify the user and then allow the user to determine whether to approve or deny the privilege as required by the application.

- **Permission Access Management**

The new model of privilege administration enhances the Android OS's default permission process. In certain cases, when a program uses a device resource, if it has received express authorization from the process of the Android OS permission framework, the improved privilege administration also needs further testing. The improved permission management requests users to determine whether to grant or deny and update the database details simultaneously. The systematic method of improved management of permission as shown in Figure 7.

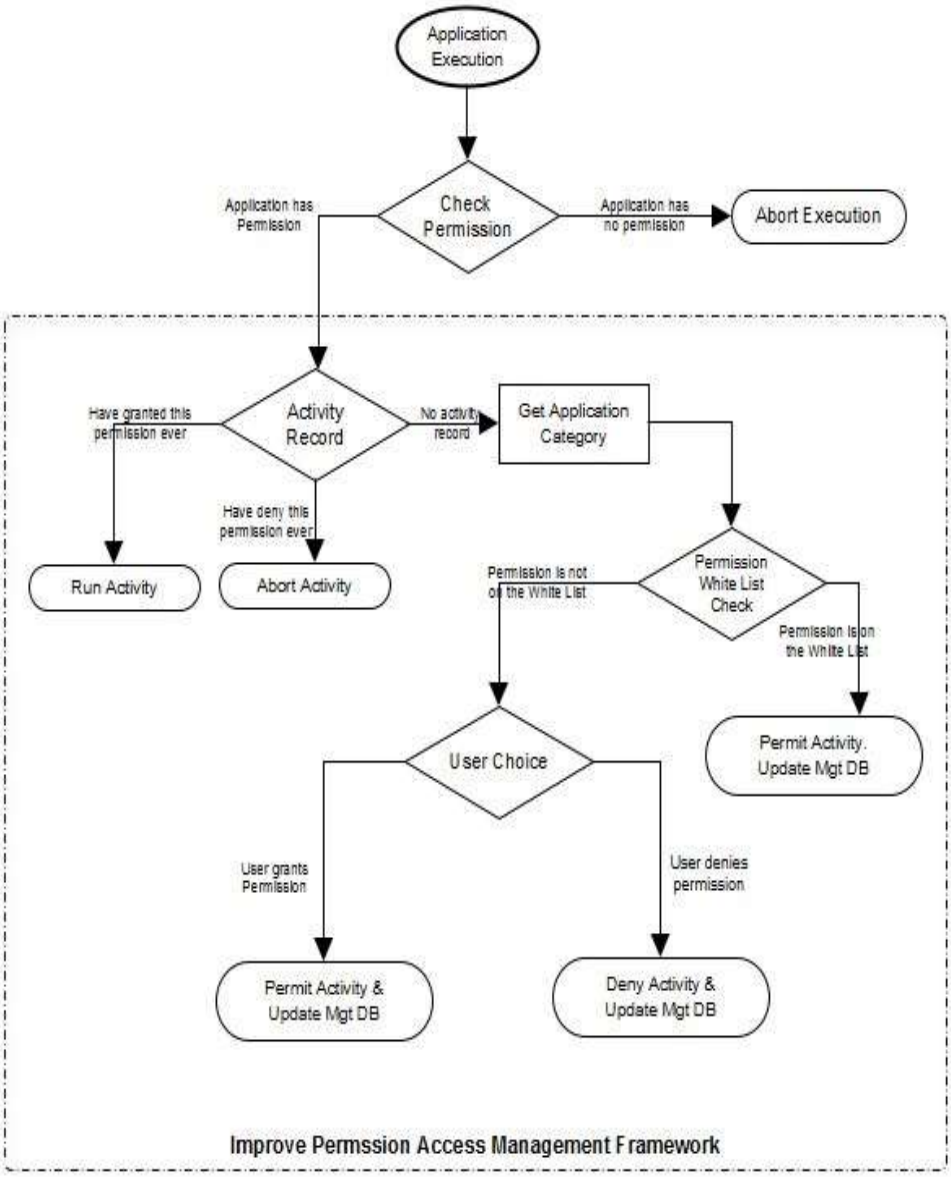


Figure 7: iPAM Framework.

VI. iPAM: TEST & RESULT DISCUSSION

- **Machine Learning Algorithm**

To get the best accurate application classifier, the most suitable machine learning algorithm needs to be chosen to train and evaluate the extracted data sets. To train the classifier, this study uses the open-source machine learning algorithms. Machine learning algorithms that are widely used include Naïve Bayes, Logical Regression, Decision Tree, and SVM. In the model, the classification features relate to the features and characteristics of the application. Nevertheless, with application development, the features and characteristics must continuously be looped and updated, so it is essential to integrate new training data to update the program classifier in advance.

However, the interaction of the program functionality has little impact on the performance of the classifier within the iPAM model. It also became evident in research before the machine learning algorithm that the logical regression method has a variety of parameter estimation method strategies, and therefore does not involve consideration of the correlation of features. Contrary to the Decision Tree and SVM, the Logical Regression algorithm makes updating the model simpler and integrates data from new features. The list of new application categories will always be updated, just like with the application classifier. Hence the logical regression algorithm is the most suitable application classifier algorithm.

- **Classifier Training Phase**

To implement the classifier for use, this study uses the Weka 3.8.4 program. Weka integrates many machine-learning algorithms capable of performing the data mining task. Data sets and sensitive API obtained from the previous steps are used as training and test data sets in Weka for the request permissions. Weka implement in machine learning algorithms such as SMO, Logical Regression, Bayesian, and Decision Tree [27]. Weka's classifier module output, as shown in Figure 8, gives the results of training and testing in text form. The Correctly Classified Instances (CCI) filed indicates that classification accuracy is up to 88.22 percent by selecting permission and the API as the machine learning feature values.

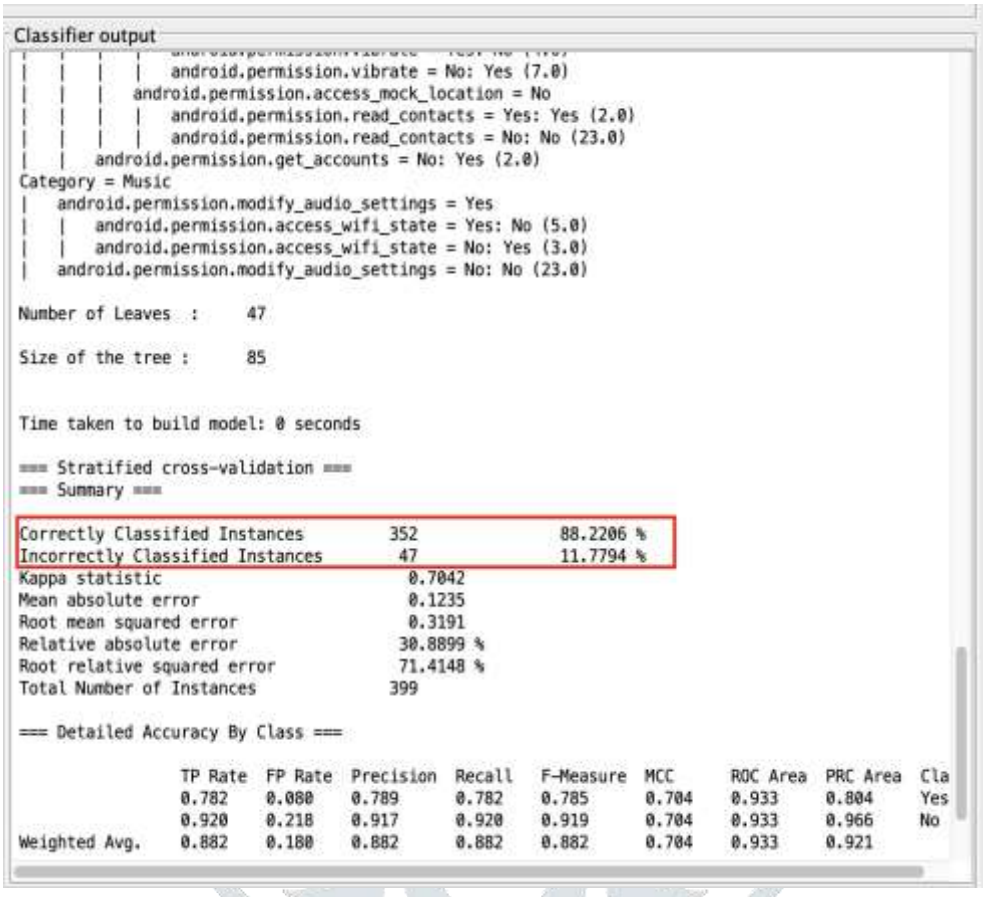


Figure 8: iPAM Classifier Module Output.

- **Method and Function Testing**

This study designs a privilege permission escalation attack to assess the system's function. The design attack package contains two applications: CatDownloader (the malicious software with no right to access SD-Card and Network) and MouseDownloader (a program user had already installed and given SD-Card and Network access permissions to it) [28], [29].

The idea behind this attack is that CatDownloader can access prohibited resources via MouseDownloader that accomplishes the privilege of escalation by CatDownloader application. The concept as illustrated in Figure 4.2. MouseDownloader simulates a download program precisely, enters the file's Uniform Resource Locator (URL), press the download button to begin downloading. The downloaded file will be stored on the SD-Card as shown in Figure 9(a).



Figure 9: (a)

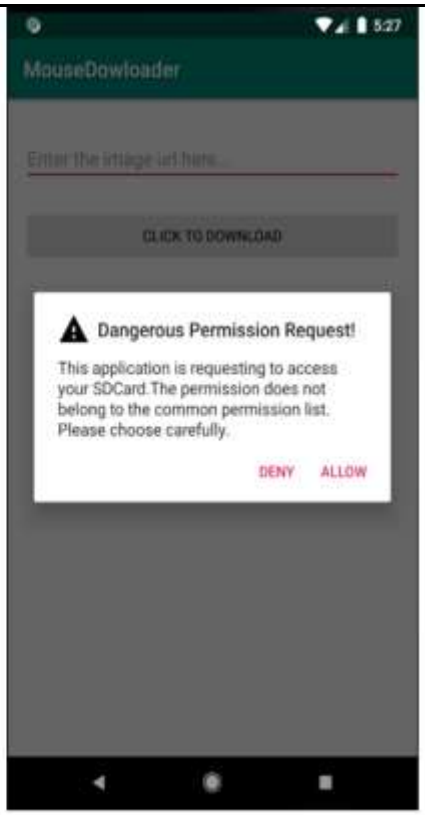


Figure 9: (b)

MouseDownloader User Interface. There are an EditText view and Button view in Figure 9(a). The user only needs to enter a valid URL and to download the image file, click on the button. The enhanced permission management system in Figure 9(b) prompts the user to have a warning dialog to notify the user of the danger of the permission requested and the option to either "Allow" or "Deny" and to secure the SD-Card data from unauthorized access.

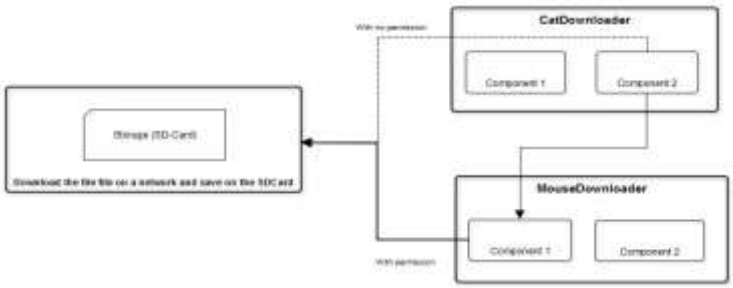


Figure 10: Privilege Escalation Attack Scenario.

In this attack as shown in Figure 10, CatDownloader invokes the exposed component through a broadcast receiver (e.g. export=true) in MouseDownloader to achieve the objective of downloading a file from the web. MouseDownloader has the privilege to download and thus save files from the web to the SD-card. It downloads files by accepting a broadcast request and then receives the URL to retrieve the file. The privilege declaration and the broadcast receiver data are specified in the AndroidManifest.xml file of the MouseDownloader application as shown in Figure 11.

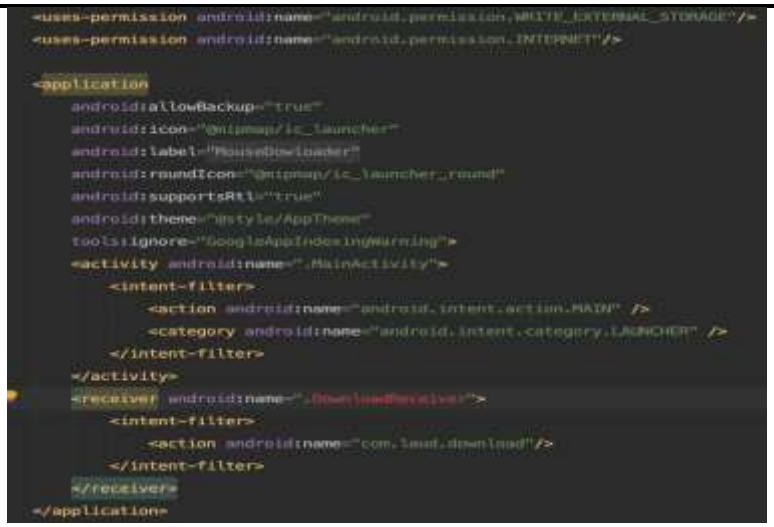


Figure 11: The AndroidManifest.xml Code Snippet of MouseDownloader.

The CatDownloader behaves as an attacker application without any permission. It is also interesting to acknowledge that, as shown in Figure 4.5, a broadcast request could also be sent to retrieve the URL file through an Intent.

```
findViewById(R.id.button_download).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setAction("com.laud.download");
        intent.putExtra( name: "url",  value: "http://someurl.com/4762yry5787.jpg");
        String filename = "myImage.png";
        intent.putExtra( name: "filename", filename);
        sendBroadcast(intent);
    }
});
```

Figure 12: Code snippet of CatDownloader.

- **Classifier Test and Simulation**

The project requires two Android Debug Bridge (ADB) software, the Android Studio IDE is currently bundled. MouseDownloader's install log, as shown in Figure 4.6. The Figure shows the application classification module tracking the application's installation procedure and extracting the feature value. The classifier then predicts the installation application category as shown in Figure 4.6. The result is that MouseDownloader belongs to the category "Tools".

```
27538-27538/? I/Xposed: [Application Classification]: new application is installing
27538-27538/? I/Xposed: [Application Classification]: PacakgeName = com.admin.downloadmanager; UID = 100085
27538-27538/? I/Xposed: [Application Classification]: Features = [0,0,0,0,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,
27538-27538/? I/Xposed: [Application Classification]: Classifier Prediction.....
27538-27538/? I/Xposed: [Application Classification]: Category Result = [tools]
27538-27538/? I/Xposed: [Application Classification]: Saving Result.....
27538-27538/? I/Xposed: [Application Classification]: classification work finished
```

Figure 13: MouseDownloader Installation Log.

- **Regular Environment**

The Trial Result. CatDownloader can successfully exploit the components of MouseDownloader to download the targeted file and save it on the SD-Card without the user's consent. The outcomes of the implementation, as shown in Figure 4.7. The file named "myImage.jpg" has been downloaded and saved successfully. Thus, the privilege escalation attack was executed by CatDownloader. The attack application completes the unauthorized execution by asking for the SD-Card and Network connection permissions. There's a toast as shown on the user interface in the trial outcome in standard Android OS environment to notify the user that the file is successfully downloaded.

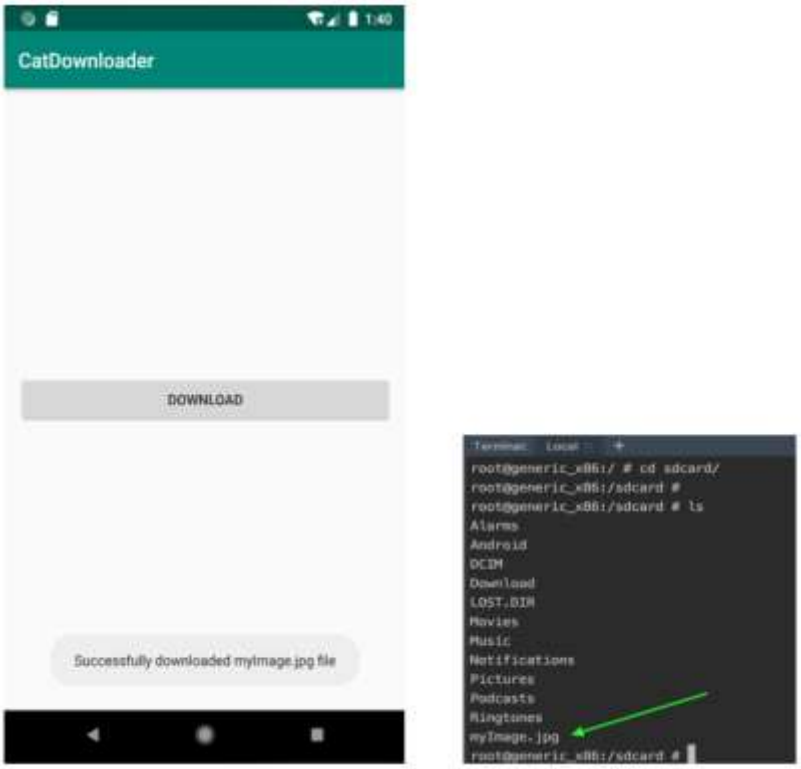


Figure 14: In Typical Android Settings, test result. The "myImage.jpg" file is revised by folder on the sd card as seen in the list

- **Model Performance Evaluation**

This segment conducts performance reviews to determine the reliability and efficacy of the model. It is evident that when new requests for permission are displayed on the device, the model will always interrupt executions. The use of RAM and Central Processing Unit (CPU) in the Android Operating System (AOS) must be tested and the time cost of the device running must be checked as well.

- **Device Resource Evaluation**

A total of three applications executed in the same test device to assess the improved model results. The CPU utilization rate for the three programs in the scenarios as shown in Table 2. The model suggests an effect on the occupancy of the CPU, but all the sequences of the uptick are no more than 9 percent, which can be ignored.

Table 1: CPU Occupancy Comparison (3 programs)

CPU Occupancy	App1	App2	App3
Without Model Deployed	35%	21%	43%
With Model Deployed	43%	24%	49%

Table 2: Memory Occupancy Comparison (3 programs)

Memory Occupancy	App1	App2	App3
Without Model Deployed	18	31	56
With Model Deployed	32	36	79

Table 3: Installation Time Comparison (3 Applications).

Installing Time	App1	App2	App3
Without Model Deployed	4155	3889	5532
With Model Deployed	4808	5139	7822

VII. CONCLUSION, RECOMMENDATION & FUTURE WORK

- **Conclusion**

In this project, iPAM is presented as proof of concept, and alternative tool to manage and administer permission as a request by applications both malign and malicious applications that seeks to exploit components of other vulnerable applications to cause harm or breach of users' privacy. The proposed model iPAM integrates machine learning and privilege administration to strengthen the default privilege framework as set out in the Android Operating System Permission Framework.

The improved model excludes unauthorized execution, such as obtaining media data from the Internet. The model proposed for implementation is based on the Xposed framework version v3.1.4. The explanation for Xposed is that, based on Android devices being rooted, this requirement creates a huge discount on Android architecture security. Translating this model into the real-world environment and on a physical Android device is important in the future.

- **Recommendations**

In addition to a more discreet approach in the practice of developers, some changes and enhancements that can be made to the Android OS to prevent accidental disclosure of device components. One of the fundamental issues is that Intents are used for both intra-component communication and inter-component interactions and that can be tricked into disclosing the application to external attacks such as permission escalation attacks that are not anticipated if developers are not sensitive and extra diligent.

Conventionally, Intent communication must be transmitted in separate channels. Similarly, usability components of applications can be isolated in such a way that, some dedicated communication channels are only for internal exported attributes to the device and exported attributes to distinct programs if required.

- **Future Work**

In future work, iPAM will be further enhanced to support automatic validation of exploited leaks through existing permission escalation attacks and handle further enhancements to analyze native code of suspected malicious applications and over privilege applications.

VIII. REFERENCES

[1] M. Zhang and H. Yin, "Automatic generation of vulnerability-specific patches for preventing component hijacking attacks," *SpringerBriefs Comput. Sci.*, no. 9783319478111, pp. 45–61, 2016, doi: 10.1007/978-3-319-47812-8_4.

[2] M. Backes, S. Gerling, C. Hammer, M. Maffei, and P. Von Styp-Rekowsky, "AppGuard - Enforcing user requirements on android apps," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7795 LNCS, pp. 543–548, 2013, doi: 10.1007/978-3-642-36742-7_39.

[3] G. S. Tuncay, S. Demetriou, K. Ganju, and C. A. Gunter, "Resolving the Predicament of Android Custom Permissions," no. February, 2018, doi: 10.14722/ndss.2018.23210.

[4] S. Fahl, M. Harbach, T. Muders, M. Smith, L. Baumgärtner, and B. Freisleben, "Why Eve and Mallory love Android: An analysis of Android SSL (in)security," *Proc. ACM Conf. Comput. Commun. Secur.*, pp. 50–61, 2012, doi: 10.1145/2382196.2382205.

[5] Z. B. Ratliff, D. R. Kuhn, and D. J. Ragsdale, "Detecting Vulnerabilities in Android Applications using Event Sequences," *Proc. - 19th IEEE Int. Conf. Softw. Qual. Reliab. Secur. QRS 2019*, pp. 159–166, 2019, doi: 10.1109/QRS.2019.00032.

[6] T. Yang, H. Cui, and S. Niu, "Dynamic loading vulnerability detection for android applications through ensemble learning," *Chinese J. Electron.*, vol. 26, no. 5, pp. 960–965, 2017, doi: 10.1049/cje.2017.07.001.

[7] A. Demontis *et al.*, "Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection," *IEEE Trans. Dependable Secur. Comput.*, vol. 16, no. 4, pp. 711–724, 2019, doi: 10.1109/TDSC.2017.2700270.

[8] S. Tarkoma, M. Siekkinen, E. Lagerspetz, and Y. U. energy consumption. Xiao, "2G, 3G, 4G, Long Term Evolution (LTE) connectivity,... - Google Scholar," 2009. https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=2G%2C+3G%2C+4G%2C+Long+Term+Evolution+%28LTE%29+connectivity%2C+touch+screen%2C+accelerometer%2C+Global+Positioning+System+%28GPS%29+and+over+1.7Ghz+of+processing+speed+and+power&btnG= (accessed Jan. 06, 2020).

[9] C. Ren, Y. Zhang, H. Xue, T. Wei, and P. Liu, "Towards Discovering and Understanding Task Hijacking in Android," *24th USENIX Secur. Symp. (USENIX Secur. 15)*, pp. 945–959, 2015, [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/ren-chuangang

[10] I. Khokhlov and L. Reznik, "Colluded Applications Vulnerabilities in Android Devices," *Proc. - 2017 IEEE 15th Int. Conf. Dependable, Auton. Secur. Comput. 2017 IEEE 15th Int. Conf. Pervasive Intell. Comput. 2017 IEEE 3rd Int. Conf. Big Data Intell. Compu*, vol. 2018-Janua, no. November 2017, pp. 462–469, 2018, doi: 10.1109/DASC-PICom-DataCom-CyberSciTec.2017.89.

[11] A. A. Sheikh, "Smartphone : Android Vs IOS," no. December, 2018, doi: 10.13140/RG.2.2.10444.46724.

[12] S. Dhama, "An Overview of Security Challenges of Android Apps Permissions," *Int. J. Inf. Comput. Technol.*, vol. 4, no. 4, pp. 373–380, 2014, [Online]. Available: http://www.ripublication.com/irph/ijict_spl/ijictv4n4spl_06.pdf

[13] T. Boksasp and U. Eivind, "Android Apps and Permissions : Security and Privacy Risks," no. June, pp. 1–143, 2012, [Online]. Available: http://www.diva-portal.org/smash/get/diva2:566356/FULLTEXT01.pdf

[14] A. Cozzette *et al.*, "Improving the security of Android inter-component communication," *Proc. 2013 IFIP/IEEE Int. Symp. Integr. Netw. Manag. IM 2013*, pp. 808–811, 2013.

[15] H. Bagheri, E. Kang, S. Malek, and D. Jackson, "Detection of design flaws in the android permission protocol through bounded verification," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9109, pp. 73–89, 2015, doi: 10.1007/978-3-319-19249-9_6.

[16] J. Han, Q. Yan, D. Gao, J. Zhou, and R. Deng, "Android or iOS for better privacy protection ?," p. 10, 2012.

[17] B. Cauchi, M. Eichelberg, A. Huewel, K. Adiloglu, H. Richter, and M. Typlt, "Hardware/software architecture for services in the hearing aid industry," 2018. doi: 10.1109/HealthCom.2018.8531138.

[18] S. Niu, R. Huang, W. Chen, and Y. Xue, "An Improved Permission Management Scheme of Android Application Based on Machine Learning," 2018, doi: 10.1155/2018/2329891.

[19] A. Bkakria, M. Graa, N. Cuppens-Boulahia, F. Cuppens, and J. L. Lanet, "Real-time detection and reaction to activity hijacking attacks in android smartphones (short paper)," *Proc. - 2017 15th Annu. Conf. Privacy, Secur. Trust. PST 2017*, pp. 253–258, 2018, doi: 10.1109/PST.2017.00037.

[20] L. Zhuo, G. Zhimin, and C. Cen, "Research on android intent security detection based on machine learning," *Proc. - 2017 4th Int. Conf. Inf. Sci. Control Eng. ICISCE 2017*, pp. 569–574, 2017, doi: 10.1109/ICISCE.2017.124.

[21] L. Onwuzurike and E. De Cristofaro, "Danger is my middle name," pp. 1–6, 2015, doi: 10.1145/2766498.2766522.

[22] D. Sounthiraraj, J. Sahs, G. Greenwood, Z. Lin, and L. Khan, "SMV-HUNTER: Large Scale, Automated Detection of SSL/TLS Man-in-the-Middle Vulnerabilities in Android Apps," no. February, pp. 23–26, 2014, doi: 10.14722/ndss.2014.23205.

[23] P. Bhiwani and C. Parekh, "Different Android Vulnerabilities," vol. 10, no. 5, pp. 1449–1455, 2017, [Online]. Available: http://www.ripublication.com

[24] R. Gurusamy and V. Subramaniam, "A Machine Learning Approach for MRI Brain Tumor Classification," 2017.

[25] G. Sidorov, A. Gelbukh, H. Gómez-Adorno, and D. Pinto, "Soft similarity and soft cosine measure: Similarity of features in vector space model," *Comput. y Sist.*, vol. 18, no. 3, pp. 491–504, 2014, doi: 10.13053/CyS-18-3-2043.

[26] Z. Ma, "Android Application Install-time Permission Validation and Run-time Malicious Pattern Detection," 2013.

[27] T. C. Sharma and M. Jain, "WEKA Approach for Comparative Study of Classification Algorithm," *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 2, no. 4, pp. 1925–1931, 2013, Accessed: Feb. 06, 2020. [Online]. Available: www.ijarcce.com

[28] L. Davi, A. Dmitrienko, A. R. Sadeghi, and M. Winandy, "Privilege escalation attacks on android," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011, vol. 6531 LNCS, pp. 346–360. doi: 10.1007/978-3-642-18178-8_30.

[29] J. Xin, Z. Wen, N. Shaozhang, and X. Yiming, *Analysis and detection of android app privilege escalation vulnerability based on machine learning*, vol. 109. Springer International Publishing, 2019. doi: 10.1007/978-3-030-03745-1_15.

- **AUTHOR PROFILE**

Frank Laud Boateng holds an MSc. Information Technology from the Kwame Nkrumah University of Science and Technology (KNUST) Kumasi, Ghana. He had his BSc. Computer Science from Kwame Nkrumah University Science and Technology (KNUST) Kumasi, Ghana. He served at the Department of Computer Science as a National Service Person where he provided tutorials to undergraduate students. He is currently a lecturer at Kessben University College, Kumasi and involved in various community service.