International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

# Detecting Malware on the Android Phones Based on Golden Jackal Optimized Support Vector Machine

**[1]Rupal Gupta, [2]Brijraj Singh Solanki, [3]Manish Kumar, [4]Murugan R.**

**Submitted:**15/04/2023 **Revised:**05/06/2023 **Accepted:**20/06/2023

**Abstract:** The Android smartphone's growth may be attributed to the phone's open-source design and high performance. Malware has been created partially because of Android's widespread use. When it comes to smartphones, Android is the most popular OS. That's why there's so much malicious software aimed at this system. Malicious software may be identified as such by analyzing its permission attributes. But this is a complex issue to solve. In this research, we use a golden jackal optimized support vector machine (GJOSVM) to classify software and evaluate whether or not it presents a threat. To achieve this goal, a dataset including 2850 sections of malicious software and 2866 sections of benign software was generated. Each piece of software in the dataset has 112 permission characteristics, and there is also a class feature that indicates whether or not the program is harmful. Each phase of the training and testing procedures used 10-fold cross-validation. The effectiveness of the models was measured using accuracy, F-1 Score, precision, and recall.

***Keywords:*** *android mobiles, detecting malware, golden jackal optimized support vector machine (GJOSVM), Android Package files (APK)*

## 1. Introduction

Malware is a significant danger to the cyber security of our nation's critical infrastructure, our service industries, and our entire civilization. Smartphones and tablets have exploded in popularity in recent years, and now they're powerful enough to replace many desktop computers' functions. Applications for usage on such gadgets have been developed by a wide range of entities, such as government and banking institutions [1]. Most of our most private and essential documents are now saved on the cloud, and we can view them from anywhere with a mobile device. Malware developers have taken note of this trend. Droid Dream, an aspect of Android malware found in over 50 apps on the official Android market, is only one example of many established incidents of Android malware. The built-in security measures of Android are essentially inadequate, and data may be leaked by even benign apps. Smartphones often serve as a repository for sensitive information, including photographs, text messages, and login passwords [2]. This makes them an easy target for bad actors. Smartphones using Google's Android OS are the market leaders. Android devices, however, account for roughly 97% of malware. The fact that practically every transaction that can be completed on a computer can now be completed on a mobile device is the driving force behind these advancements, and it is also the fundamental reason why they have occurred. At the very beginning of choice for mobile devices are the dimensions, as well as the benefits that come with the measurements at the point where transportation is concerned [3]. As processing capabilities have increased and as interest in gadgets has grown, it has gained a lot of traction among attackers. Attackers will almost always target sites that are of very high interest to a large number of people. In the digital age, if a specific person or organization is not targeted, the objective is to spread the malware to the most significant number of users possible. When looking at the mobile OSs that are used all over the globe, the two that stand out as the most popular are Android and iOS [4]. Other OSs have a consumption rate that is just 1.7% of the total, in contrast to these two OSs, which account for 98.3% of the entire usage in the globe. The number one spot goes to the Android OS, which has a use rate of 70 percent, while the number two spot goes to the iOS OS, which has a rate of 28.3 percent. Machine learning algorithms have become particularly significant in areas where antivirus software is inadequate to guarantee that mobile platforms, which can run software that is becoming more complicated, may become better protected against harmful software [5]. This is because mobile

*1Assistant Professor, College of Computing Science and Information Technology, Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India, Email id: r4rupal@yahoo.com*

*2Assistant professor, School of Computer Science & System, JAIPUR NAITONAL UNIVERSITY, JAIPUR, India, Email Id: brijraj.solanki@jnujaipur.ac.in*

*3Assistant Professor, School of Engineering and Computer, Dev Bhoomi Uttarakhand University, Uttarakhand, India, Email Id: ce.manish@dbuu.ac.in*

*4Associate Professor, Department of Computer Science and IT, Jain(Deemed-to-be University), Bangalore-27, India, Email Id: murugan@jainuniversity.ac.in*

platforms can run software that is becoming increasingly complex. Based on authorizations and API requests, and even though these methods gather conversations between applications on smartphones and Android systems, any interaction that occurs within the applicable limitations isn't considered in the assessment. As a result, it is not enough to detect malicious software that is not asking for suspicious resources. Malware assaults have so spread to mobile devices, making it imperative that we take measures to secure our mobile infrastructure [6]. In this work, a golden jackal-optimized support vector machine is employed to find malicious Android phone attacks. The results of our experiments prove that our technique successfully identifies Android malware.

The rest of the sections of the paper are structured as follows. In Section 2, we give a literature review on similar efforts in Android malware detection. Our detection system, including the feature extraction method, is described fully in Section 3. In Section 4, we detail the experiment and its findings. Finally, we conclude the task in Section 5.

## 2. Related work

They build the link between the permission and API using a machine learning technique to detect malware by mining the patterns of approval and API function calls obtained and utilized by Android applications. Despite this, the harmful samples they have gathered are not sufficient [7]. A framework for feature-based learning that focuses on the behaviors of requested permissions and API requests and that applies the SVM, Decision Tree, and Bagging algorithms. However, they only extract authorization and API as features, which leads to a poor level of accuracy since they only remove a few different kinds of characteristics [8]. Despite this, research on mobile malware is still in its infant stage. The methods that are now available to identify mobile malware and other flaws in security have varied degrees of both strengths and drawbacks [9]. The random forest algorithm uses three distinct feature selection methods. The effects of implementing three alternative feature selection methods effective, high weight and effective group feature selection are evaluated. Applying feature selection approaches improves accuracy regarding metrics and needed time, according to experiments on the Drebin dataset [10]. They employed an evolutionary algorithm to find Android smartphone malware. They contrasted our system with several cutting-edge algorithms to assess it. Finally, their suggested strategy can detect zero-day malware [11]. Numerous ML-based methods for detecting malware on Android have been presented. Multiple difficulties arise from ignorance of the technologies available for detecting malware on Android devices [12]. Additionally, to ensure device compatibility, authorization, and hardware characteristics are simultaneously stated in the manifest file of an application (app). To characterize applications, we extract permissions, API requests, and hardware characteristics [13]. The TANMAD algorithm, a two-step Android malware detection method, first narrows the potential malware families to be detected before using sub-graph isomorphism matching. The modeling of object reference information by creating directed graphs, or ORGB, is the main innovation of their study [14].

## 3. Methodology

The main goal is to create a classifier that, for the most part, classifies training data as positive and only labels training or testing data as negative when it sufficiently deviates from training data. Given that innocuous Android apps are far simpler to find than malicious ones, it will be the perfect choice for our purposes. The antivirus software business known as Zeman gave us the files that are included in the dataset so that we may do this research. To extract functionality from pre-packaged Android apps (APKs), we use an open-source project known as Androguard. Our study's suggested flow is shown in figure 1.

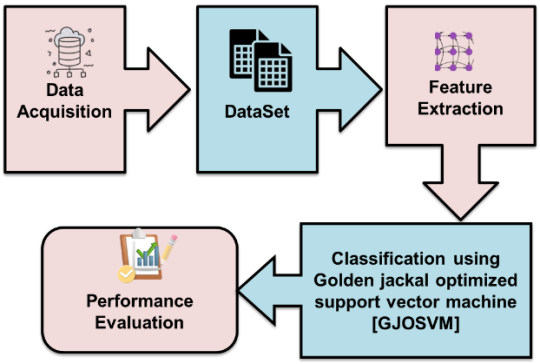


**Fig. 1.** Flow of GJOSVM

### 3.1 Data Acquisition and Dataset

There are 2870 legitimate apps and 2854 malicious ones in the newly produced dataset. Virus Total and Virus Share have verified the presence of harmful and non-malicious Android apps in this dataset [15]. This information cannot be downloaded by any user, which defeats the purpose of the infection. Due to the sensitive nature of the work that will be performed via the use of the corporate e-mail account, specific authorization for harmful software has been secured by corresponding with the proprietors of this website through e-mail. The whole collection consists entirely of binary information. A value of 0 for any of the 117 characteristics implies that the program does not seek authorization for the feature in question, whereas a value of 1 indicates that it does. A value of 0 for this class attribute implies that the program is safe, whereas a value

of one indicates that it is malicious. The apps' access controls are shown in Table 1.

**Table 1.** Features of Android APK authorization

| | | | |
|---|---|---|---|
| internet | use_credentials | keep_screen_on | write_external_storage |
| mount_format_filesystems | broadcast_sms | read_owner_data | read_phone_state |
| mount_unmount_filesystems | read_sync_settings | write_sync_settings | get_tasks |
| access_fine_location | access_all_downloads | write_internal_storage | camera |
| flashlight | baidu_location_service | write_contacts | vibrate |
| wake_lock | battery_status | broadcast_wap_pushf | disable_keyguard |
| get_accounts | expand_status_bar | write_apn_settings | change_network_state |
| battery_stats | ua_data | modify_phone_state | bluetooth |
| restart_packages | access_coarse_updates | read_messaging_extension_data | nfc |
| bind_accessibility_service | send_messages | bind_remoteviews | get_package_size |
| kill_background_processes | interact_across_users_full | install_packages | bluetooth_admin |
| raised_thread_priority | authenticate_accounts | set_debug_app | access_superuser |
| bind_input_method | network_state | clear_app_cache | broadcast_wap_push |
| reorder_tasks | get_contacts | send_respond_via_message | receive_wap_push |
| persistent_activity | wake_lockc | read_logs | cd_messagel |
| access_location_extra_com. | manage_accounts | write_secure_settings | vibrategcmgs |
| read_contacts | flashlightghardware_c. | write_call_log | receive_boot_completed |
| receive_sms | change_wifi_state | read_datagpersonal_infol | access_wifi_state |
| read_sms | write_media_storage | delete_packages | send_sms |
| call_phone | bind_wallpapertntb | cd_message | system_alert_window |
| write_sms | connectivity_internal | hardware_test | set_wallpaper |
| set_wallpaper_hints | read_call_log | location | modify_audio_settings |
| record_audio | access_mock_location | change_wifi_multicast_state | read_calendar |
| broadcast_sticky | signal_persistent_processes | ua_datal | bind_vpn_service |
| maps_receivel | internetgcmg | bind_print_service | bind_notification_listener_service |
| process_outgoing_calls | change_configuration | receive_user_present | bind_device_admin |
| bind_wallpaper | read_profile | write_profile | access_network_state |
| read_external_storage | receive_mms | network | read_settings |
| write_settings | set_activity_watcher | access_download_manager | access_coarse_location |

### 3.2 Feature Extraction

To successfully implement any kernel, we must first isolate the most crucial aspects of the application. APK files, the standard format for Android application packaging, are quite similar to ordinary Java jar files. To process these files and extract features, we use the open-source software Androguard. Androguard has a user interface that is not too complicated and may be used to do analysis and reverse engineering on Android apps.

Every APK requires a manifest file that, among other things, requests authorization to access specific protected components of the Android OS. Accessibility for various hardware devices, sensitive aspects of the OS, and specific sensitive features of other programs are all included in these pieces. For instance, the "android.permission.INTERNET" permission demands the ability to access the Internet, but the "android.permission.READ CONTACTS" permission desires the ability to read the mobile contacts database of the user.

After we have obtained the list of rights that are being sought, we split it into two categories: built-in permissions that are considered standard and permissions that are not regarded as standard. We build a binary vector for legal permissions, and each item corresponds to a built-in permission. The entry is given a value of 1 if the program wants that permission and a value of 0 if it does not seek that permission. In the case of non-standard permissions, we break the strings up into three sections: the prefix (which is often "com" or "org"), the portion containing the organization and product, and the permission name itself. We don't pay attention to any instances of the phrases "android" or "permission" since they are so common.

### 3.3 k-Fold Cross-validation

The technique for segmenting the data set that is utilized for training the model through executing the calculation procedures of the models in classification processes is called k-fold cross-validation. In addition to these assessments, cross-validation is an additional method that may be used to evaluate learning algorithms. This method involves segmenting the data and comparing the segments to one another. In the process of cross-validation, the training set and the validation set are repeatedly combined so that each data point gets the opportunity to be verified. The k-fold cross-validation method is the fundamental kind of cross-validation. K-fold cross-validation is the popular technique for model selection and error estimates of classifiers. This is mainly attributable to the fact that it is both versatile and useful when used in data mining applications. The k-fold cross-validation approach utilized in this work is shown essentially in figure 2. Figure 2 shows the results of a k value is 10 cross-validation. The dataset is partitioned into k pieces in this approach. Iteratively, models are trained on k-1 parts and tested on a single sample. When k iterations are performed, k outputs are generated. Simply averaging the gathered findings will give you the average metrics of the models' performance.

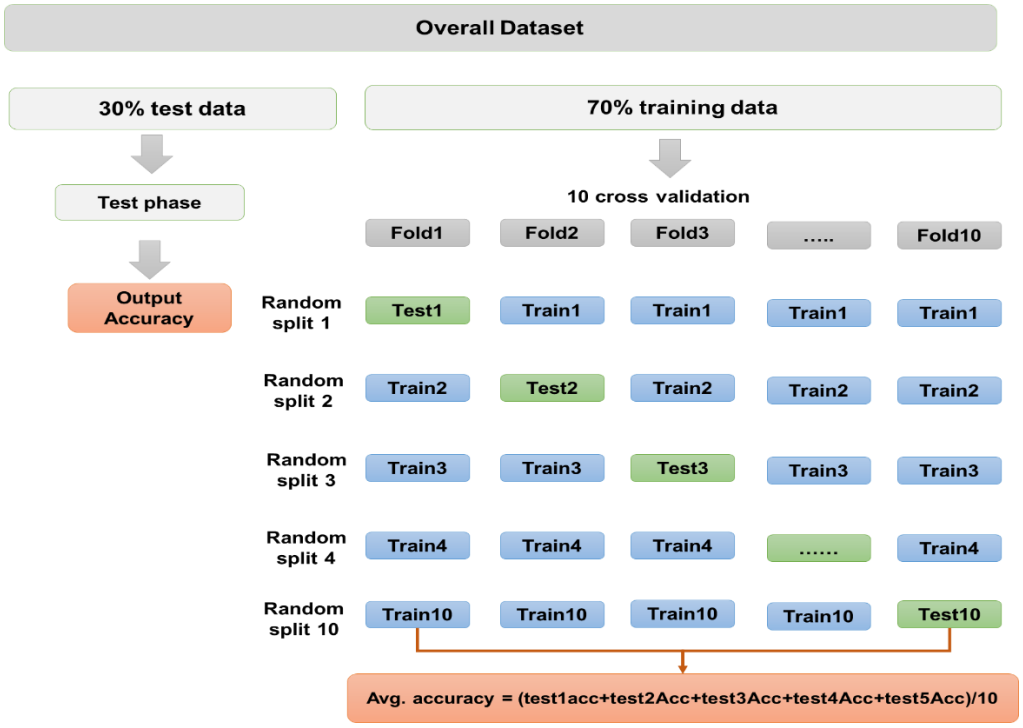


**Fig. 2.** k-fold cross validation

### 3.4 Classification using GJOSVM

Inspired by the strategies used by golden jackals, a type of canid found in many regions of the globe, Golden Jackal Optimization (GJO) is a natural-based optimization method. Metaheuristic algorithms, like GJO, are problem-solving techniques that take natural cues. The GJO algorithm is based on many behaviors seen in the golden jackal. Jackals, for instance, have earned a reputation for versatility because of their capacity to thrive in habitats as varied as deserts and woodlands. In addition to being formidable foes, they use shrewd methods of hunting. Define the viral detection challenge by considering the viruses' properties, the data at hand, and the processes already in use. It's up to you to figure out what goes into it and how you'll judge success. Produce an initial set of solutions representing possible detection procedures. Several factors, including genetic markers, machine learning algorithms, and feature extraction methods, may affect the accuracy of a detection system. Therefore there are many possible solutions. Evaluate the viability and performance of each option in the population. Here, we put the parameters and characteristics we've chosen for viral detection through their paces and assess how well they function by calculating metrics like detection accuracy, sensitivity, specificity, and false positive rate. Explore possible solutions by simulating the hunting behavior of golden jackals. Solutions, which stand in for detection strategies, may be improved by adjusting their parameters or characteristics in light of the collective knowledge gathered from other solutions. This may need trying out different feature sets, tweaking algorithm settings, or changing which genetic markers are used. The fitness measurements and convergence criteria used to evaluate the performance of GJO and other algorithms are just as important as the quality and variety of the original population.

For detecting purposes, we use the SVM algorithm, whose defining characteristic is its foundation in structural risk reduction. Optimize learning's generalizability or the extent to which a small training set can ensure a sizeable independent test set that maintains a small error. Little sequences determined by regular access to the unusually short series in the sample might include normal intermittent, causing the SVM classifier to produce classification error; thus, the implementation of a detection module, which provided follows the level of risk using malware to make decisions. Taking into mind the fact that the impact of malicious software on a smartphone's operating system and its user is distinct from the damages brought about by the introduction of a risk factor (also known as RF or Risk Factor), RF is applied to every one of the system's brief sequences. Act of malice committed with the intention of providing a weight, the correct basis. If the behavior of the system and user poses a more significant security danger, which results in an RF that is more than 1, the value is set to 1. The introduction of risk (also known as Risk Rank or RR) is a program that serves as a measure of the quantitative detection of malware. The RR is defined as follows:

$$RR = \sum_{i=1}^{n} ncs_i \times RF_{ncsi} \quad (1)$$

Establish a malware detection threshold known as D, whose value will be based on the experiment's findings. Based on our findings, a D value of 17 will serve as the most effective detection threshold. When the RR is found to be more than the D that was computed, the program is finally identified as malicious software.

## 4. Result and Discussion

Classifications have been carried out using the Android Malware Dataset's 112 characteristics to determine if apps are malicious or not. Experiments used a computer that had an Intel® i5® 10200H CPU operating at 2.40 GHz, an NVIDIA GTX1650Ti GPU, and 24 GB of RAM. In the approach of cross-validation used for training, the classification models were decided to have a value of k equal to 10. While training the models, each of the 116 inputs and the single class feature was used, together with all of the characteristics included in the dataset. Performance metrics are used to analyze and compare the performance of various suggested models. Metrics like "accuracy, precision, recall, and the F-1 Score" were used in this investigation. While training the proposed model, the kernel function was analyzed and found to be an RBF (Radial Basis Function) with a numerical tolerance of 0.0010 and an iteration of 100.

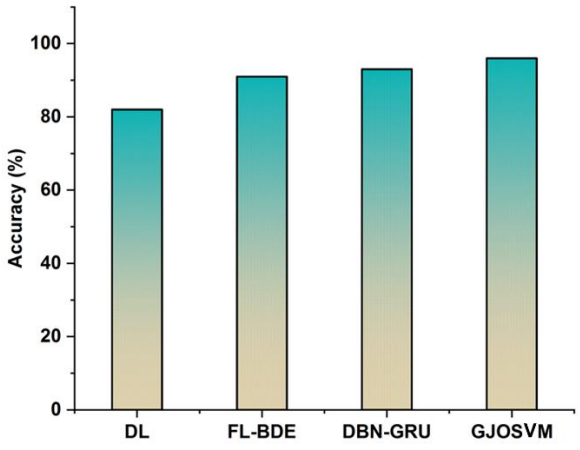


**Fig.3.** Accuracy

Figure 3 shows the accuracy of the proposed system. The accuracy of malware detection systems can vary depending on various factors, including the techniques and algorithms used, the quality and diversity of the dataset, and the sophistication of the malware samples. Achieving high accuracy is an ongoing challenge due to the constantly evolving nature of malware. DL has attained 82 %, FL-BDE has acquired 91 %, DBN-GRU has reached 93 %, and the proposed system achieves 96 % accuracy. It shows that the proposed approach has more effective than the existing one. The Equation (2) for determining accuracy is as follows:

$$Accuracy = (True\ Positives\ True\ Negatives) / (True\ Positives\ True\ Negatives + False\ Positives + False\ Negatives) \quad (2)$$

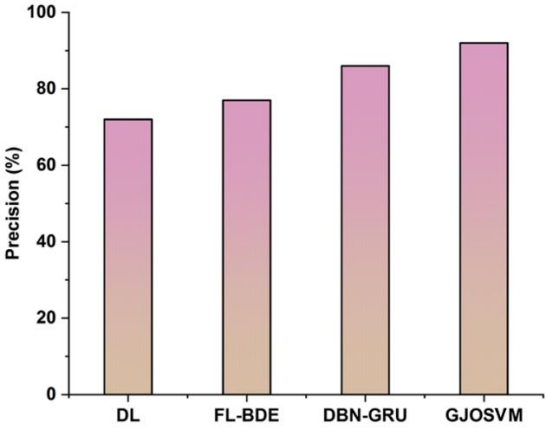


**Fig. 4.** Precision

Figure 4 shows the precision of the proposed system. The accuracy of a malware detection system is measured by how many malicious samples out of a whole set are really malicious. It is a test of how well positive malware predictions can be made. The equation (3) for determining precision is as follows:

$$Precision = True\ Positives / (True\ Positives + False\ Positives) \quad (3)$$

Accurately recognizing malware while not incorrectly labeling safe files as harmful is only possible with a system that has a low false positive rate, which is shown by high precision. In security-sensitive contexts, where false positives may have dire repercussions, accuracy in malware detection systems is a crucial parameter. DL has attained 72 %, FL-BDE has acquired 77 %, and DBN-GRU has reached 86 %, whereas the proposed method achieves 92 % of precision. It shows that the proposed approach has more effective than the existing one.

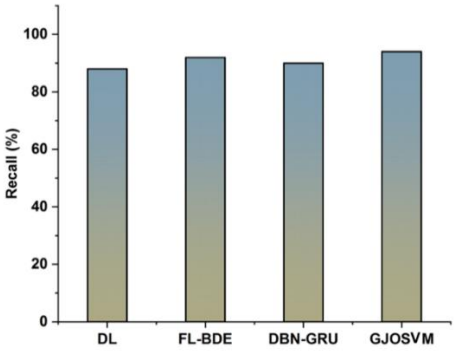


**Fig. 5.** Recall

Figure 5 shows the recall of the proposed system. The recall metric calculates the percentage of malware samples properly recognized relative to the overall malware samples in the dataset. This metric is also known as sensitivity or actual positive rate. It measures how well a malware detection system can detect malware samples. The Equation (4) for the recall is as follows:

$$Recall = True\ Positives\ /(True\ Positives + False\ Negatives) \quad (4)$$

It measures how well a system can detect malware in a dataset. DL has attained 88 %, FL-BDE has acquired 92 %, DBN-GRU has reached 90 %, whereas the proposed method achieves 94 % of recall. It demonstrates that the suggested technique is more successful than the existing one.

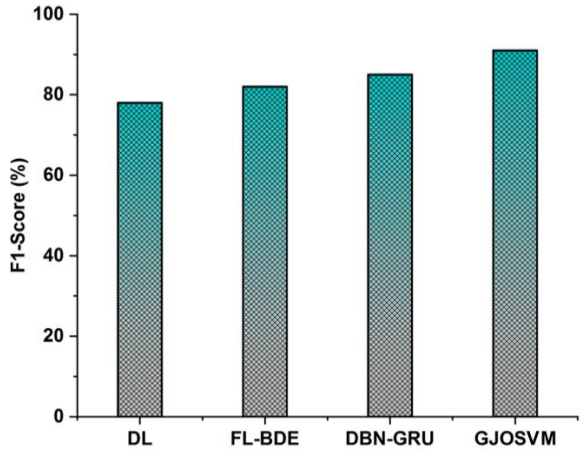


**Fig. 6.** F1-Score

A balanced evaluation of a malware detection system's performance is presented by the F1-score, a statistic that combines accuracy and recalls into a single number. It takes into account both the accuracy with which malware samples may be detected as well as the capacity to detect all instances of malware. The F1-score calculation equation (5) is as follows:

$$F1 - score = 2 * (Precision * Recall) / (Precision + Recall) \quad (5)$$

The F1-score ranges from 0 to 1, with 0 denoting subpar performance and 1 signifying flawless accuracy and recall. DL has attained 78 %, FL-BDE has acquired 82 %, and DBN-GRU has reached 85 %, whereas the proposed system attains 91 % of the f1-score. It shows that the proposed approach has more effective than the existing one.

## 5. Conclusion

In this work, classification procedures were carried out utilizing GJOSVM techniques utilizing the data of 2850 harmful apps and 2866 non-malicious applications used in the Android OS. To conduct in-depth performance assessments, 10-fold cross-validation was used. The suggested model has values of 96% for accuracy, 92% for precision, 94% for recall, and 91% for F-1 Score. These values are all much higher than the average. It is feasible to create antivirus software that is more effective by using the models that are advised for the identification of malware. In subsequent investigations, the dataset will be enlarged, and various machine-learning approaches will be used to identify malware.

## References


[1] Mbunge, E., Muchemwa, B., Batani, J. and Mbuyisa, N., 2023. A review of deep learning models to detect malware in Android applications. Cyber Security and Applications, p.100014.

[2] Mijwil, M.M., 2020. Malware Detection in Android OS Using Machine Learning Techniques. Data Science and Applications, 3(2), pp.5-9.

[3] Awais, M., Tariq, M.A., Iqbal, J. and Masood, Y., 2023, February. Anti-Ant Framework for Android Malware Detection and Prevention Using Supervised Learning. In 2023 4th International Conference on Advancements in Computational Sciences (ICACS) (pp. 1-5). IEEE.

[4] Wang, Z., Liu, Q. and Chi, Y., 2020. Review of Android malware detection based on deep learning. IEEE Access, 8, pp.181102-181126.

[5] Bayazit, E.C., Sahingoz, O.K. and Dogan, B., 2020, June. Malware detection in Android systems with traditional machine learning models: a survey. In 2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA) (pp. 1-8). IEEE.

[6] Ehsan, A., Catal, C. and Mishra, A., 2022. Detecting Malware by Analyzing App Permissions on Android Platform: A Systematic Literature Review. Sensors, 22(20), p.7928.

[7] Yanamadni, V. R. ., Seetha, J. ., Kumar, T. S. ., Kannaiah, S. K. ., J, B. ., & Brahmaiah, M. . (2023). Computer-Aided Detection of Skin Cancer Detection from Lesion Images via Deep-Learning Techniques. International Journal on Recent and Innovation Trends in Computing and Communication, 11(2s), 293–302. https://doi.org/10.17762/ijritcc.v11i2s.6158

[8] Wang, X. and Li, C., 2021. Android malware detection through machine learning on kernel task structures. Neurocomputing, 435, pp.126-150.

[9] Muzaffar, A., Hassen, H.R., Lones, M.A. and Zantout, H., 2022. An in-depth review of machine learning based android malware detection. Computers & Security, p.102833.

[10] Niveditha, V.R. and Ananthan, T.V., 2019. Detection of Malware attacks in smart phones using Machine Learning. International Journal of Innovative Technology and Exploring Engineering, 9(1).

[11] Keyvanpour, M.R., Barani Shirzad, M. and Heydarian, F., 2023. Android malware detection applying feature selection techniques and machine learning. Multimedia Tools and Applications, 82(6), pp.9517-9531.

[12] Dhabliya, P. D. . (2020). Multispectral Image Analysis Using Feature Extraction with Classification for Agricultural Crop Cultivation Based On 4G Wireless IOT Networks. Research Journal of Computer Systems and Engineering, 1(1), 01–05. Retrieved from https://technicaljournals.org/RJCSE/index.php/journal/article/view/10

[13] Waheed, W.F. and Alyasiri, H., 2023. Evolving trees for detecting android malware using evolutionary learning. International Journal of Nonlinear Analysis and Applications, 14(1), pp.753-761.

[14] Shimpi, P.M. and Pise, N.N., 2023, March. An Empirical Study of Efficient Malware Detection Analysis on Android Mobile Phones using Machine Learning. In 2023 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS) (pp. 1520-1526). IEEE.

[15] Zhu, H.J., Gu, W., Wang, L.M., Xu, Z.C. and Sheng, V.S., 2023. Android malware detection based on multi-head squeeze-and-excitation residual network. Expert Systems with Applications, 212, p.118705.

[16] Zhang, W., Wang, H., He, H. and Liu, P., 2020. DAMBA: Detecting android malware by ORGB analysis. IEEE Transactions on Reliability, 69(1), pp.55-69.

[17] Yilmaz, A.B., Taspinar, Y.S. and Koklu, M., 2022. Classification of Malicious Android Applications Using Naive Bayes and Support Vector Machine Algorithms. International Journal of Intelligent Systems and Applications in Engineering, 10(2), pp.269-274.

[18] Almomani, I., Alkhayer, A. and El-Shafai, W., 2022. An automated vision-based deep learning model for efficient detection of android malware attacks. IEEE Access, 10, pp.2700-2720.

[19] Atacak, İ., 2023. An Ensemble Approach Based on Fuzzy Logic Using Machine Learning Classifiers for Android Malware Detection. Applied Sciences, 13(3), p.1484.

[20] Lu, T., Du, Y., Ouyang, L., Chen, Q. and Wang, X., 2020. Android malware detection based on a hybrid deep learning model. Security and Communication Networks, 2020, pp.1-11.