


# Supervised Learning Methods Comparison for Android Malware Detection Based on System Calls Referring to ARM (32-bit/EABI) Table

Rinanza Zulmy Alhamri <sup>1, </sup>, Toga Aldila Cinderatama <sup>2, </sup>, Kunti Eliyen <sup>3\*, </sup>, and Abidatul Izzah <sup>4, </sup>

<sup>1,2,3,4</sup> Department of Information Technology, Politeknik Negeri Malang, Indonesia

\* Corresponding author: [kunti.eliyen@polinema.ac.id](mailto:kunti.eliyen@polinema.ac.id)

Received: 12 February 2024

Accepted: 21 March 2024

Revised: 13 March 2024

Available online: 20 June 2024

**To cite this article:** Alhamri, R. Z., Cinderatama, T. A., Eliyen, K., & Izzah, A. (2024). Supervised Learning Methods Comparison for Android Malware Detection Based on System Calls Referring to ARM (32-bit/EABI) Table. *Journal of Information Technology and Cyber Security*. 2(1), 15-24. <https://doi.org/10.30996/jitcs.10511>

## Abstract

Android malware detection research is a topic that is still being developed. From all the detection techniques developed, dynamic analysis methods have become interesting because they trace the suspect application system calls. Based on the system calls, by utilizing machine learning, the suspect application can be classified as malware or benign. Comparing the machine learning methods is important to determine what method is best to support malware detection. This article aims to explain more clearly and simply the way to conduct Android malware detection based on system calls step by step using classification. Furthermore, it presents the system calls sequence conversion referring to the arm(32-bit/EABI) table, which has 398 system calls (0-397) as features. It will provide a comparison of several supervised machine-learning methods for classifying Android applications. This initial research is part of the other research that has the purpose of developing a malware detection system based on an Android application. This research can be used to develop the best machine learning to classify malware applications using a Support Vector Machine (SVM), Decision Tree (DT), K-Nearest Neighbour (KNN), and Naive Bayes (NB). The result can be concluded that the KNN method has the lowest performance in detecting Android malware apps, with an accuracy of only 0.50. In comparison, the NB method has an accuracy of only 0.69. SVM and DT models have similar accuracy and recall results of 0.79 and 0.75, respectively, but DT obtained higher precision and scores of 0.83 and 0.76, respectively. Although in this study, the classification performance of DT is better than SVM, based on comparison with the results of previous research, SVM is a suitable method for Android malware detection based on system calls. It is proven by the results of research comparisons that the SVM method is always the method with the highest accuracy score among other methods. For the next research, the SVM method can be used to develop a malware detection system for Android applications.

**Keywords:** Android, Decision Tree, Machine Learning, Malware Detection, Supervised Learning, Naive Bayes.

## 1. Introduction

Malicious software or malware is malicious software criminals develop to weaken the host's hardware. Currently, there are many malware applications with various impacts for Android applications especially. One of the impacts often encountered and often becomes a user issue is the decreased performance of the device itself, such as a device that runs much slower than before. This is because malware running in the background causes high CPU (Central Processing Unit) load, low storage capacity, and more significant memory usage (Negi et al., 2021; Selvaganapathy et al., 2021). Malware has several types, and their respective effects include adware, which creates lots of popup ads; spyware, which can monitor user activity on mobile devices; ransomware, which can lock data; and trojans, which attach to official applications but, when installed, can damage the system (Chandini et al., 2019).

Many previous studies have been carried out to develop malware detection for Android applications in Android mobile devices. Both developing methods for detecting and implementing detection applications. Much research has been carried out in terms of implementing detection methods, such as using the Support Vector Machine (SVM) method based on system call activity from applications suspected of being malware (Akbi et al., 2018). They provided the performance of SVM to classify the Android apps based on its system calls. Furthermore, malware detection has been carried out using SVM classification, which has been improved using the Decision Tree (DT) method to increase detection accuracy based on the condition of the Android mobile device (Yang et al., 2020). In the same way, they provided the performance of SVM that improved using DT to classify Android applications based on its system calls, Application Programming Interface (API), and permission called Dalvik Instruction. K-Nearest Neighbour (KNN) and Naive Bayes (NB) method also have been conducted to classify Android applications using static or dynamic detection based on requested permissions, system API calls, and the activity (Arslan & Yurttakal, 2020; Pang & Bian, 2019).

Meanwhile, in terms of implementing the detection application, researchers have also conducted various studies to apply malware detection methods to Android mobile devices. The majority of malware detection applications are implemented as agent applications to collect data on the device, then the data is sent to a server which acts as a detection model. Malware detection based on activities and conditions in applications and systems is called dynamic detection, while malware detection that examines the program contents of suspected applications is called static detection (Bhatia & Kaushal, 2017; Malik, 2019). A static malware detection application based on Android has been implemented as an agent (Habibi et al., 2017; Jusoh et al., 2021). As a static detection, the implementation was more complex than dynamic detection. The static detection depends with fixed system and (Habibi et al., 2017) utilized Kaspersky system to implement static detection. Then the application of dynamic malware detection applications has been developed with a local client-server topology, both detections based on device resource conditions (Ribeiro et al., 2020) and based on system call activity (Zhang et al., 2022). Also hybrid method can be used to detect Android malware applications by integrating the features after static and dynamic analyses (Dhalaria & Gandotra, 2021).

Researchers have also compared the performance of each machine-learning method in detecting malware. Each performance depends on what data researchers use to classify. The machine learning method can support malware detection of an Android application, both based on system call activity from the application and the condition of the Android device. A comparison of machine learning methods has been carried out for Android malware classification based on system calls (Anshori et al., 2019). The methods compared include SVM, NB, DT, Random Forest, Log Regression, and KNN. It used 19 attributes, consist of 18 features only and 1 class category. The result said that Random Forest method had the highest accuracy results with a percentage of 76%. Ribeiro et al. (2020) compared the machine learning methods based on condition of the Android device. The dataset was collected manually and the result said that SVM was the best method with 99.83% accuracy using proposed Host-based Intrusion Detection System. Hadiprakoso et al. (2022) also compared the machine learning method based on the condition of the Android device using DREBIN dataset. It resulted SVM was the best method with 96.94% accuracy. Hybrid analysis also have been carried out by combining static detection like condition of device and dynamic detection like API call command using several deep learning methods (Hadiprakoso et al., 2021). The result said that Long Short-Term Memory (LSTM) model was the best method with 98.7% accuracy. Pure dynamic malware detection using Android device system call has been carried out by comparing several machine learning methods (Zhang et al., 2022). The result said that Multi Layer Perceptron (MLP) was the best method with 99.34% accuracy.

Comparing the machine learning methods is important to determine what method is best to support malware detection. This research was conducted by referring Zhang et al. (2022) where utilizing Android applications system call as the subject to be analyzed. This article focuses to explain more clearly and simpler the way to conduct the Android malware detection based on system call step by step. Furthermore, this article presents the system calls sequence conversion referring to the arm(32-bit/EABI) table, which has 398 system calls (0-397) as features. This is the novelty of this research as a first step before creating a larger system for implementing and deploying Android-based malware detection applications. For sure, it will be provided the comparison of several supervised machine learning methods for classifying the Android application. The purpose of this research is to provide evidence of the best supervised machine learning method for detecting Android malware based on its system calls. This article is part of the research to develop Android malware detection system that utilizing machine learning with Python based as server, cloud database Firebase as data intermediary, and Android application as a client interface. Later then, the

result will be used to determine what supervised machine learning methods that best to support the Android malware detection system.

## 2. Methods

The development of supervised learning machines using the Python programming language where Python has several libraries to developing machine learning. Development is carried out in some sub stages including Data Collection, Model Implementation, and Model Testing as shown in the Fig. 1.

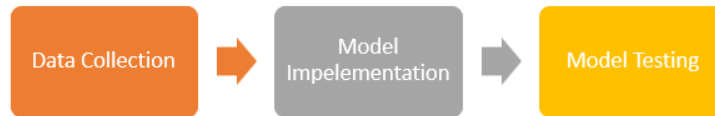


Fig. 1. Research method.

### 2.1. Data collection

The purpose of data collection is to collect system call data from Android apps as a dataset. Referring to Zhang et al. (2022), it is needed 3000 system call per apps to get the best result. To validate the benign or malware apps then it is used [virustotal.com](https://www.virustotal.com) to give label to the state of Android apps. The data collection consists of 50 benign apps and 50 malware apps. To make the data collection safer and more efficient, then it is used Android virtualization using Genymotion and Android Studio emulator. To produce natural 3000 system calls from the Android application, then it is used automatic and random stress-test application using The Monkey.

How to produce the dataset for detecting Android malware as shown in Fig. 1 is based on some Android malware detection frameworks. Manzil and S (2023) stated in DynaMalDroid framework, dataset conducted from utilizing strace command and Monkey tool of the benign or malware Android application. As well as that, Shakya and Dave (2022) stated that data collection used strace command to achieve the system calls and MonkeyRunner for conditioning the Android device run the apps automatically. Data collection was conducted to obtain a dataset. The dataset in question contains system-call sequence data from Android applications (both benign and malware), which have been converted into a series of numbers. Fig. 2 shows the stages in creating a dataset to develop a machine learning model.



Fig. 2. Data collection stages.

Based on Fig. 2, the following is a detailed explanation about dataset creation.

#### 1) Application Determination

Benign applications are easy to obtain by visiting Google Play, which has been verified for security. Meanwhile, malware applications were obtained from information on the internet and then verified as unsafe by checking the .app files on [virustotal.com](https://www.virustotal.com). If the results of checking the application file are verified as dangerous (a warning displayed in red), then the application is considered a malware application.

#### 2) Taking System Calls

The applications that were classified as benign or malware all had their system-call data taken while they were running on an Android device. A simulation environment with the Genymotion and Android Studio emulator is used to retrieve system-call data. The following are detailed steps for retrieving system-call data for a sample application.

- The .app application is installed on the Genymotion and Android Studio emulator device, then runs the application.
- In the adb shell terminal running on the host operating system, in this case, Ubuntu 22.04.3, the shell command searches for the PID and package name of the application.

#### 3) Once the package name is known, the application in question will be given random commands automatically (random stressed-test) using the Monkey application.

#### 4) As soon as the stressed test command is executed, the strace shell command is activated on the PID of the application in question, and the strace results are converted to a txt format file. Strace was carried out for one minute, which resulted in more than 3000 documented system calls.

#### 5) The results of the strace in the form of a .txt file are stored in the Android device emulator so that they can be retrieved to be stored on the host PC.



ing was developed to classify Android applications as benign or malware based on system call data. The following are detailed steps in developing supervised machine learning using the SVM, DT, KNN, and NB method until model accuracy testing is carried out.

1) Reading Dataset

The .csv format dataset contains system call sequences. Using the Pandas library, the .csv format file can be read, and you can choose which columns to read.

2) Splitting Dataset

The number of samples in the dataset divided into training data and testing data. In this study, 80% was used for training, while 20% was used for testing. Ratio 80/20 for splitting data into training and testing is common division. Furthermore based on Gholamy et al. (2018) ratio 80/20 can avoid overfitting and overestimate accuracy empirically. This section also differentiates between reading the dataset in the label column and the system call column.

3) Feature Extraction

Feature extraction is applied to assign a token to each system call number in the dataset (tokenization). In this research, the CountVectorizer method is used, which can transform system call sequences into vector data (vectorization). Apart from providing token values and vector transformations, the countvectorizer also counts the number of occurrences of system call numbers in the dataset. By carrying out feature extraction, it will help improve the performance accuracy of the model implementation.

4) Machine Learning Model

The application of the machine learning model used the SVM, DT, KNN, and NB. In the Scikit-learn library, the SVM uses SVC method.

- SVM - This research uses a SVM linear kernel type because the classification is carried out in binary. It was used default hyperparameters tuning including C = 1.0, Degree = 3, and Gamma using scale type.
- DT - Whereas the DT used DecisionTreeClassifier method. Default hyperparameters was used for tuning including Criterion = 'gini' and Splitter = 'best'. KNN - Then the KNN used KNeighborsClassifier method. It was used default hyperparameters also including n\_neighbors = 5, Weight using uniform type, and Algorithm = 'auto'. Below is the KneighborsClassifier method used in the program.
- NB – The last method used NB method with GaussianNB. NB method used default hyperparameters tuning including Priors = 'None' and var\_smoothing = 1e-9. Below is the GaussianNB method used in the program.

### 2.3. Model testing

Model testing aims to obtain the performance of each machine learning method for classifying the data. Still by using scikit-learning library, the model testing can be done. Table 1 below is used to explain various terms for evaluating classification in this malware detection cases.

**Table 1**

Example of malware detection evaluation.

No. Application (app)	Real Label	Classification Result
1	Benign	Benign
2	Benign	Benign
3	Benign	Benign
4	Benign	Benign
5	Benign	Malware
6	Malware	Benign
7	Malware	Malware
8	Malware	Malware
9	Malware	Malware
10	Malware	Malware

From Table 1 can be obtained:

- True Positive: Malware classification results are according to the label = 4,
- True Negative: Benign classification results are according to the label = 4,
- False Positive: Malware classification results are not according to the label = 1, and
- False Negative: Benign classification results are not according to the label = 1.

Based on the terms, the testing includes:

- Accuracy

Accuracy is the percentage of labels that the model successfully predicted. If the model can classify 8 applications accurately from 10 data applications then accuracy (A) is 0.8 as shown in Eq. (1),

$$A = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

where TP is True Positive, TN is True Negative, FP is False Positive, and FN is False Negative. A high accuracy score means more accurate the model to classify Android applications whether mal-ware or benign application generally

- Precision

Precision measures the ratio between true positive prediction and all positive result prediction. If the model can classify 5 data applications into malware from 10 data applications, but only 4 data applications that definitely malware (true positive) and 1 data application that turned out to be benign (false positive) then the precision (P) is 0.8 as shown in Eq. (2),

$$P = \frac{TP}{TP+FP} \quad (2)$$

where TP is True Positive and FP is False Positive. A high precision score means more accurate the model to classify the data applications into malware (positive) result.

- Recall

Recall or sensitivity measures the ratio between true positive prediction and all positive data. If the dataset consists 5 malware data applications from 10 data applications and the model can classify 4 data applications into definitely malware applications (true positive). Then there are 1 other data applications that classifying into benign applications that turned out to be malware (false negative) then the recall (R) is 0.8 as shown in Eq. (3),

$$R = \frac{2 \times (R \times P)}{R + P} \quad (3)$$

where TP is True Positive and FN is False Negative. A high recall score means more correctly the model to predict malware applications (positive) instance.

- F1Score

F1Score is a comparison of the weighted average precision and recall that concludes a harmonic means. F1Score formulas is shown in the Eq. (4),

$$F1Score = \frac{2 \times (R \times P)}{R + P} \quad (4)$$

where R is Recall and P is Precision. A high F1Score score means more correctly model to predict malware applications (positive) result minority in not balance data.

### 3. Results and Discussion

The dataset consists of 50 benign applications and 50 malware applications, where each application system-call sequence consists of 3000 sequences. Table 2 is top benign and malware Android application that was used to be dataset. The number sequence of system calls refers to the ID/NR on the Android operating system using the ARM architecture processor in the arm(32-bit/EABI) table, which has 398 system calls (0-397) as features. The number of samples in the dataset divided into training data and testing data. In this study, 80% was used for training, while 20% was used for testing.

For obtaining the performance of some models, the testing will be conducted. Model testing includes score of accuracy, precision, recall, and F1score. Accuracy testing is carried out by calculating the percentage of correct test results based on the labels on the test data as shown in Eq. (1). In the Scikit-learn library, the accuracy score() function is used to obtain accuracy values. The performance result of all model implementation is shown in Table 3.

**Table 2**  
Top 10 Benign and Malware Android Application Packages (APK).

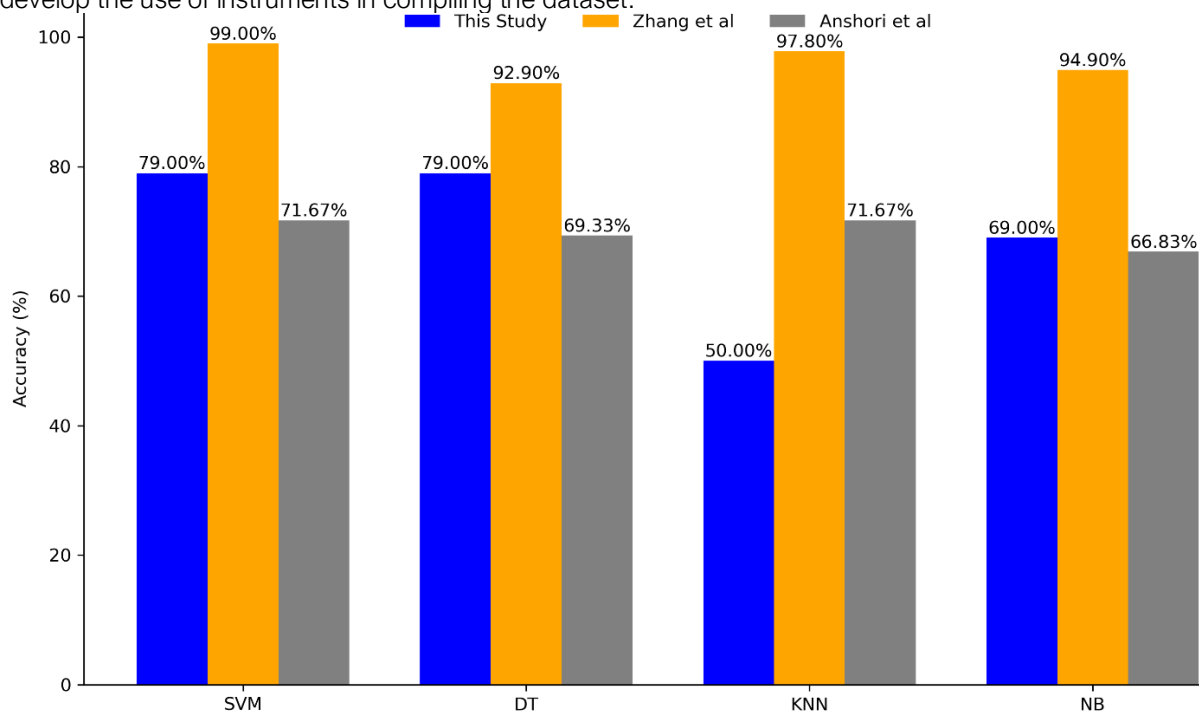
No	Benign	Malware
1	Tool Box	Auto Photo Blur
2	AlQuran	Cool Emoji
3	Sektch Book	Phone Cleaner
4	Smart SMS	Secure VPN
5	Smile Emoji	Super Battery
6	Al Kitab	Theme Message
7	Custom Keyboard	Hyper Cleaner
8	Battery Calibration	Horoscope
9	App Lock	Same Launcher
10	Universal PDF	TikTok18

**Table 3**  
Performance comparison of model implementation.

No.	Model	Accuracy	Precision	Recall	F1 Score
1	SVM	<b>0.79</b>	0.78	<b>0.79</b>	0.75
2	DT	<b>0.79</b>	<b>0.83</b>	<b>0.79</b>	<b>0.76</b>
3	KNN	0.50	0.49	0.50	0.49
4	NB	0.69	0.65	0.68	0.65

After implementing data collection according to the research plan, there are many findings that need to be discussed. The first finding was the use of the strace command on an Android device which actually required root access, so the Android device had to be rooted. Rooting the device is too risky, so data retrieval has to be done on the Android emulator. However, using the Android Emulator to retrieve system calls data is not very effective because many of the latest applications fail to install on the Android Emulator. The second finding was about system calls sequence condition. After retrieving system calls sequence data from strace command to each Android applications, there was a lot of string with the val-ue "<...", "-- SIGSEGV", "+++ exited", and etc. that doesn't match with the system calls data referring to the arm(32-bit/EABI) table. 3000 sequences of system calls data retrieval from each application had to be eliminated a lot due to inappropriate data strings.

Based on the findings stated previously, there are potential limitations that can be used as a reference for improving subsequent research. They can influence the training results of the machine learning so that the classification accuracy results obtained are less than optimal. First, retrieval of system calls data used the Android emulator, so the application installed as a data sample was limited. This allows the dataset not to be formed optimally. It is recommended that future research be able to use actual Android devices by being able to anticipate the device rooting process. Second, method for converting system calls data into a sequence of numbers was done by taking 3000 lines and then cleaning out the strings that do not match. It is recommended to take more than 3000 data system calls so that when the data is cleaned the data sequence is not reduced much or methodically first clean up strings that do not match and then take a number of lines. And the last limitation is this research had a limited time period, making it difficult to develop the use of instruments in compiling the dataset.



**Fig. 6.** Accuracy score comparison with previous research.

From the performance result of model implementation, it could be clearly concluded from this case that KNN had the lowest performance. Based on Table 3, KNN method had accuracy, precision, recall, and F1score of 0.50, 0.49, 0.50, and 0.49 respectively. Then NB method had accuracy, precision, recall, and F1score of 0.69, 0.65, 0.68, and 0.65 respectively. SVM and DT had similar results in accuracy where the score was 0.79. The score Recall between SVM and DT also had similar result of 0.79. The difference between them was the result of precision and F1score, where DT had the higher result then SVM. SVM

obtained precision and F1score of 0.78 and 0.75 respectively, meanwhile DT obtained 0.83 and 0.76. It means that DT was able to predict positive results or benign Android applications more accurate than SVM.

By obtaining classification test results and their comparisons, these results need to be compared with the results of previous research. In this article, it will compare the classification accuracy results with the accuracy of previous malware detection research which purely used dynamic analysis in the form of system calls including (Anshori et al., 2019) and (Zhang et al., 2022). Fig. 6 is a comparison of accuracy score of this research with previous research based on four methods including SVM, DT, KNN, and NB where in general the various methods used by (Zhang et al., 2022) have high accuracy with the highest accuracy score being the SVM method with a score reaching 99.0%. Furthermore based on table, the highest accuracy score in (Anshori et al., 2019) are in the SVM and K-NN method at 71.67%. While the highest accuracy score in this study were SVM and DT with a percentage of 79.0%.

From the results of comparison with previous research, this study has an accuracy score that is not as good as (Zhang et al., 2022) because the dataset is not optimally formed due to the system calls data not fully amounting to 3000 strings. However, the results of the classification accuracy score from several methods exceed (Anshori et al., 2019) except KNN method because this study uses of features refers to arm(32-bit/EABI) table with a total of 398 features. In general, the results of the classification accuracy score from the three studies that being compared, SVM is the method with the highest accuracy based on the three studies that have been carried out. Therefore, SVM method is suitable for classifying Android malware with dynamic analysis based on system calls.

#### 4. Conclusions

This initial research is part of the other research that has a purpose to develop a malware detection system for Android application. From the research that has been conducted, it can be concluded that the KNN method has the lowest performance to detect Android malware applications with accuracy only 0.50. SVM and DT model have similar accuracy and recall result of 0.79 and 0.75 respectively, but DT obtained higher precision and F1score of 0.83 and 0.76 respectively. Although in this study the classification performance of DT is better than SVM, based on comparison with the results of previous research, SVM is a suitable method for Android malware detection based on system calls data. It is proven by the results of research comparisons that the SVM method is always the method with the highest accuracy score among other methods. For the next research SVM method can be used to develop a malware detection system for Android application. Suggestions that can be given for the next research include increasing accuracy and expanding the dataset. Furthermore data collection must be taken from the running applications in real device to give result in relevant environment, not just an Android emulator.

#### 5. CRediT Authorship Contribution Statement

**Rinanza Zulmy Alhamri:** Conceptualization, Data curation, Formal Analysis, Funding acquisition, Investigation, and Methodology. **Toga Aldila Cinderatama:** Project administration and Writing – original draft. **Kunti Eliyen:** Resources, Software, Visualization, and Writing – review & editing. **Abida-tul Izzah:** Supervision and validation.

#### 6. Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

#### 7. Acknowledgments

The authors would like to thank the anonymous referees for their helpful comments and suggestions.

#### 8. Data Availability

System calls list of the ARM(32-bit/EABI) table available in <https://chromium.googlesource.com/chromiumos/docs/+master/constants/syscalls.md>

#### 9. Funding

This research is funded by Politeknik Negeri Malang, Indonesia with DIPA fund number SP DIPA 023.18.2.677606/2023 and Grant number 5625/PL2.1/HK/2023.

---



## 10. References

- Akbi, D. R., Herlambang, S., Basuki, S., & Sari, Z. (2018). Deteksi Malware Android Berdasarkan System Call Menggunakan Algoritma Support Vector Machine. *Seminar Nasional Teknologi Dan Rekayasa (SENTRA)*.
- Anshori, M., Mar'i, F., & Bachtiar, F. A. (2019). Comparison of Machine Learning Methods for Android Malicious Software Classification based on System Call. *2019 International Conference on Sustainable Information Engineering and Technology (SIET)*, 343–348. <https://doi.org/10.1109/SIET48054.2019.8985998>
- Arslan, R. S., & Yurttakal, A. H. (2020). K-Nearest Neighbour Classifier Usage for Permission Based Malware Detection in Android. *Icontech Journal of Innovative Surveys, Engineering & Technology*, 4(2), 15–27. <https://doi.org/10.46291/ICONTECHvol4iss2pp15-27>
- Bhatia, T., & Kaushal, R. (2017, June). Malware detection in android based on dynamic analysis. *2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security)*. <https://doi.org/10.1109/CyberSecPODS.2017.8074847>
- Chandini, S. B., Rajendra, A. B., & Nitin, S. G. (2019). A Research on Different Types of Malware and Detection Techniques. *International Journal of Recent Technology and Engineering*, 8(2S8), 1792–1797. <https://doi.org/10.35940/ijrte.B1155.0882S819>
- Dhalaria, M., & Gandotra, E. (2021). A Hybrid Approach for Android Malware Detection and Family Classification. *International Journal of Interactive Multimedia and Artificial Intelligence*, 6(6), 174–188. <https://doi.org/10.9781/ijimai.2020.09.001>
- Gholamy, A., Kreinovich, V., & Kosheleva, O. (2018). Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation. *International Journal of Intelligent Technologies and Applied Statistics*, 11(2), 105–111. [https://doi.org/10.6148/IJITAS.201806\\_11\(2\).0003](https://doi.org/10.6148/IJITAS.201806_11(2).0003)
- Habibi, M., Ismail, S. J., & Sularsa, A. (2017). Implementation of Malware Detection Service on Android. *E-Proceedings of Applied Science*, 3(3), 1839–1847.
- Hadiprakoso, R. B., Aditya, W. R., & Pramitha, F. N. (2022). Analisis Statis Deteksi Malware Android Menggunakan Algoritma Supervised Machine Learning. *Cyber Security Dan Forensik Digital*, 5(1), 1–5. <https://doi.org/10.14421/csecurity.2022.5.1.3116>
- Hadiprakoso, R. B., Qomariasih, N., & Yasa, R. N. (2021). Identifikasi Malware Android Menggunakan Pendekatan Analisis Hibrid dengan Deep Learning. *Jurnal Teknologi Informasi Universitas Lambung Mangkurat*, 6(2), 77–84. <https://doi.org/10.20527/jtiulm.v6i2.82>
- Jusoh, R., Firdaus, A., Anwar, S., Osman, M. Z., Darmawan, M. F., & Razak, M. F. A. (2021). Malware detection using static analysis in Android: a review of FeCO (features, classification, and obfuscation). *PeerJ. Computer Science*, 7. <https://doi.org/10.7717/peerj-cs.522>
- Malik, S. (2019). Anomaly based Intrusion Detection in Android Mobiles: A Review. *International Journal of Engineering Research and Technology*, 8(10), 698–710. [www.ijert.org](http://www.ijert.org)
- Manzil, H. H. R., & S, M. N. (2023, December 28). DynaMalDroid: Dynamic Analysis-Based Detection Framework for Android Malware Using Machine Learning Techniques. *2022 International Conference on Knowledge Engineering and Communication Systems (ICKES)*. <https://doi.org/10.1109/ICKES56523.2022.10060106>
- Negi, C., Mishra, P., Chaudhary, P., & Vardhan, H. (2021). A Review and Case Study on Android Malware: Threat Model, Attacks, Techniques and Tools. *Journal of Cyber Security and Mobility*, 10(1), 231–260. <https://doi.org/10.13052/jcsm2245-1439.1018>
- Pang, J., & Bian, J. (2019). Android Malware Detection Based on Naive Bayes. *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, 10, 483–486. <https://doi.org/10.1109/ICSESS47205.2019.9040796>
- Ribeiro, J., Saghezchi, F. B., Mantas, G., Rodriguez, J., Shepherd, S. J., & Abd-Alhameed, R. A. (2020). An Autonomous Host-Based Intrusion Detection System for Android Mobile Devices. *Mobile Networks and Applications*, 25, 164–172. <https://doi.org/10.1007/s11036-019-01220-y>
- Selvaganapathy, S., Sadasivam, S., & Ravi, V. (2021). A Review on Android Malware: Attacks, Countermeasures and Challenges Ahead. *Journal of Cyber Security and Mobility*, 10(1), 177–230. <https://doi.org/10.13052/jcsm2245-1439.1017>
- Shakya, S., & Dave, M. (2022). *Analysis, Detection, and Classification of Android Malware using System Calls*. <https://doi.org/10.48550/arXiv.2208.06130>
- Yang, M., Chen, X., Luo, Y., & Zhang, H. (2020). An Android Malware Detection Model Based on DT-SVM. *Security and Communication Networks*. <https://doi.org/10.1155/2020/8841233>

Zhang, X., Mathur, A., Zhao, L., Rahmat, S., Niyaz, Q., Javaid, A., & Yang, X. (2022). An Early Detection of Android Malware Using System Calls based Machine Learning Model. *Proceedings of the 17th International Conference on Availability, Reliability and Security*, 1–9. <https://doi.org/10.1145/3538969.3544413>

