National College of Ireland

MSc Cyber Security

Research Project

Rohith Satheesh Kumar

Student ID: X19207611

School of Computing

National College of Ireland

Supervisor: Imran Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	ROHITH SATHEESH KUMAR
Student ID:	X19207611
Programme:	MSc Cyber Security **Year:** 2021-2022
Module:	Research project
Lecturer:	Imran Khan
Submission Due Date:	15/08/2022
Project Title:	INTRUSION DETECTION OF KERNEL-ROOTKITS IN ANDROID DEVICES USING MACHINE LEARNING– RANDOM FOREST
Word count: 4500	**Page: 23**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ROHITH SATHEESH KUMAR

Date: 15/08/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS

AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Abstract:

Due to the prevalence of sensitive information on Smartphones, rootkits provide a major risk. Rootkits on Android-based smartphones have access to features that aren't accessible on PCs, such as GPS, the battery, and the microphone and speaker. This makes them particularly dangerous. Smartphone users are at greater danger of infection as open source and unlicensed third-party platforms and applications are being used. A kernel-level rootkit's threat to Android's operating system is also examined in depth in this study, as is the potential use of a system call to discriminate between calls from a regular app and those coming from an infected one.

Automated rootkit detection is possible using random forest, an approach that relies on prior data. For the purpose of detecting the rootkit, it takes use of datasets produced by feeding data and collecting system calls from infected and non-infected operating systems. A root kit identification algorithm for Android-based systems is trained using this dataset. **(javatpoint, 2019)**

Keywords: Kernel rootkits, Android, Random forest technique, Machine learning**.**

TABLE OF CONTENTS:

1. INTRODUCTION:

Smartphones are one of those technological marvels whose ubiquity is only certain to grow. This amount is predicted to climb to 7,690 million by 2027, according to Statista, which estimates that there are now 6,259 million smartphone users worldwide**.(• *Smartphone subscriptions worldwide 2027 | Statista*, no date)** Since more people are using smartphones, there's a higher chance of becoming infected with malware like a rootkit or some other kind of cyber attack. When it comes to GPS and other location-based services, smartphones have a greater number of hardware and software implementations than do desktop or laptop computers. Customized versions of the Linux Kernel platform are used in many Android handsets, and they comprise millions of Java lines. There have been 50 times as many malware breakouts on Android smartphones as there have been on iOS devices; hence the project is primarily focused on Android. According to online statistics, 19 dangerous programmes were available for download via Google Play. **(Bojan Jovanović, 2021)**

According to data from 2019 for the most frequent form of malware, Trojans accounted for 93.93% of all infections. **(• *Distribution of Android malware 2019 | Statista*, no date)**Trojan horses utilize false or deceptive identities to fool their victims into thinking they are dealing with something else entirely. Some of the most harmful Trojan horses are Back orifice, Rootkit, and Beast Trojan. Hackers may use Rootkits to get access to computers and its software without permission. An attacker with root or Administrator privileges may also install rootkits. Rootkits come in a wide variety of forms, but Kernel rootkits operate at the very heart of the operating system, giving them complete control. Having Rootkit as the most common Trojan horse, and Trojan malware as the most common form of malware, both of these assaults might be quite devastating to an Android device. Lookout's security labs recently discovered a Rooting Malware called "AbstarctEmu," which was distributed through the Google Play store along with 19 utility applications. When a user opens an infected programme, the virus is launched.**(Lab, 2021)** Exploits like CVE-2020-0041, CVE-2020-0069 and CVE-2019-2215 are used by the malware in order to exploit vulnerabilities. **(Ranajit Singh Mehroke, 2019)**

This research focuses on Android Kernel-Rootkits utilizing prior knowledge and understanding of rootkits and the threats they bring. The Linux kernel, on which Android is based, contains the bulk of the operating system's security measures. Driver management, power

management, memory management, device management, and resource access are all handled by the Linux kernel. As a result, a rootkit that manages to infect the Linux kernel has complete control over the operating system. Rootkits at the kernel level may be detected using "Random forest," a machine learning approach that is both succinct and versatile. In order to develop a rootkit detection method, the study takes use of datasets acquired through various system calls. **Analyzing data sets that contain both continuous and categorical variables will be done using the Random Forest Algorithm. Using it to solve classification issues resulted in better outcomes than other methods.**

2. LITERATURE REVIEW:

This section of the paper describes how similar work was conducted in past studies; previous publications relate to rootkit cyberattacks against the Linux-based Android operating system. Researchers are always looking for novel approaches to detect rootkits, as seen by the aforementioned articles. Researchers are increasingly using Machine learning and deep learning Automation to identify Rootkits because of their high accuracy and ability to learn from the data they acquire. Because ML-based techniques may be improved in terms of effectiveness by tweaking their algorithms and collecting new data, they are utilized.

- **Machine learning based approach:**

Kernel rootkit research is heavily influenced by the latest trend in machine learning, which allows for training and learning from the data that is presented to it. Machine learning algorithms have been demonstrated in several experiments to automatically detect both known and unknown malware. System calls and API calls are used in combination with machine learning research to discover rootkit. The Levenberg-Marquardt algorithm, behavior-based algorithms, and fully convolution neural networks are among the many machine learning and deep learning methods they use (FCN).

(Singh *et al.*, 2017)Malware was detected using a combination of hardware performance counters, machine learning algorithms based on signatures, and Scikit-learning, a free open source machine learning library for the Python programming language. According to reports, the outcome was a resounding 98% affirmative. It was claimed that a machine learning algorithm may be used to analyse system call times and determine the best course of action. **(Luckett,**

Todd McDonald and Dawson, 2016)(Levenberg Marquardt algorithm). By comparing infected and non-infected systems and identifying the resulting system calls as abnormal or non-anomalous, rootkits were discovered. The system correctly categorized 67.7 percent of the system calls it was fed, and recognized 82% of them when fed a single call. Rootkit signatures were detected using conventional machine learning techniques by **(Sayadi *et al.*, 2021)**and StealthMiner, a novel machine learning-based approach for detecting potential stealthy malware tracks at run-time via commands based on fully convolutional neural networks. The StealthMiner is said to be 6.52 times faster than this system, which has a detection rate of 94%. Machine learning algorithms can be used to identify Kernel-level rootkits, according to a few others.**(Kuzminykh and Yevdokymenko, 2019; Nadim, Lee and Akopian, 2021)** The majority of methods suggested one of the following future studies proposing the use of a superior Machine Learning technology and applying the strategy on mobile devices.

But each of these methods had some challenges. For example, the Levenberg-Marquardt Machine Learning algorithm was used as a reasonable alternative to the signature-based detection method. However, this algorithm will take a long time to converge if the model has more than 10 parameters, and the process only used a small number of datasets to study rootkit detection. Second, most of the methods are based on Windows/Desktop and use a static method to evaluate the issue.

- **Virtual Machine/Hypervisor based work:**

This kind of research requires the creation of a virtual or sandbox environment in order to discover malware. Several rootkit detection applications and memory forensics use behavior-based algorithms and machine learning for the detection mechanism. Memory forensics is included in the area of "digital forensics," which collects and presents digital evidence for cybercriminal investigations. Memory forensics has been examined extensively as a method for identifying malicious activity in the memory of a computer system.

A proposal by **(Bickford, Ganapathy and Iftode, 2012)** involves the use of virtual machines (VMs), memory forensics, machine learning methods, and rootkit detection tools such as Patagonix (which verifies the code's integrity) and Gibraltar (which checks the integrity of the kernel). The primary objective of this study is to be energy-conscious while using rootkit

protection measures. This paper aimed to create a balance between a strong defence strategy and an efficient energy solution. Rootkits may be detected using a static analysis approach based on virtual machines (VMs) running on a hypervisor **(Xie and Wang, 2013)**At the hypervisor level, the suggested rootkit detection method uses deep information extraction and cross-verification. **(Tian *et al.*, 2019)** employed Virtualization and Machine learning approaches with different Machine algorithms to compare which produces a more accurate result in identifying the Rootkit. Additionally, VMM was used to extract the kernel module's run-time behaviours and to build the feature vectors.

The primary concern with the suggested models above is that VMs themselves might be the target of an assault to disrupt the detection systems. The paper by **(Bickford *et al.*, 2010; Bickford, Ganapathy and Iftode, 2012)** does not indicate the accuracy with which Rootkits may be identified. It consisted mostly of excerpts from two separate books by various authors. Other disadvantage from **(Xie and Wang, 2013)**is that it was a static technique of detection; it also focuses on extending the idea to other hypervisors and some advanced kernel rootkits may leverage the VM-aware ways to recognize the hardware-assisted virtualization environment. **(Tian *et al.*, 2019)**In addition, the majority of models are evaluated against the Windows environment.

- **IoT and Hardware based:**

It is essential to protect IoT devices against Vulnerabilities such as Malware, since they are becoming more widespread in a variety of settings. Before and during the epidemic, there was a 700-fold increase in IoT malware.

(Jiang, Lora and Chattopadhyay, 2020)suggested LDRDet, a Trusted Execution Environment-based method for detecting kernel rootkits in IoT devices. This detection is based on system calls and requires monitoring hardware events continually for abnormalities. The framework was successfully tested against four unique kinds of malware. Another approach **(Nagy *et al.*, 2021)**was to use an IoT-based trusted execution environment (TEE) such as ARM-based embedded boards, which are typically supported by the vast majority of IoT devices. It monitors system call hooking and searches for irregularities. Combining external peripherals such as PCB and JTAGa with memory forensics allowed **(Guri *et al.*, 2015)** to effectively detect rootkits at

the Android kernel level. In this instance, the hardware retrieves and reconstructs a particular area of the kernel's memory for further examination.

The disadvantage of these papers is that they only address IoT devices, and the one on Android devices needs an additional component to be linked to the smartphone in order to detect malware, which is neither feasible nor practical for the end user. In addition, the LKRDet framework cannot identify kernel-level rootkits that manipulate the HPC value.

- **Systems call Hooking:**

By intercepting a system call, it is possible to manipulate data sent between user-space applications and the operating system. Following this, two investigations were conducted, one using system call hooking in combination with a dynamic programme slicing methodology and the other using a signature-based approach in conjunction with a conventional probing method. The dynamic programme slicing approach took use of kernel hooks and generated a HookMap to calculate the number of kernel hooks that may be exploited for resistance. This approach has the problem of relying on incorrect dynamic slicing, while other linear programming methods provide more precise dynamic slicing in an appropriate amount of space and time.**(Wang *et al.*, 2008; Brodbeck, 2012)**

Research Niche:

Related works	Strengths	Limitation
(Singh *et al.*, 2017)	High level of Accuracy	Windows based
(Luckett, Todd McDonald and Dawson, 2016)	Use of Neural Network and High level of accuracy	The algorithm used was very slow to converge and size of dataset handled was very small
(Sayadi *et al.*, 2021)	Used various deep and machine learning algorithm to provide high level of accuracy	Limited to Windows based computers
(Bickford, Ganapathy and Iftode, 2012)	Energy aware approach along with rootkit detection	Doesn’t propose anything new with the rootkit detection and doesn’t state anything about the level of accuary the model produce
(Tian *et al.*, 2019)	Uses both Virtualization and ML to generate high level of accuracy.	Limited to Windows based computers

(Jiang, Lora and Chattopadhyay, 2020)	Tested against four different types of rootkits	The framework used has its own limitation
(Nagy *et al.*, 2021)	The approach followed works best with most the IoT based devices	Limited to IoT devices
(Guri *et al.*, 2015)	Uses external PCB to detect the rootkits which can overcome the disadvantage of of VMs	Uses external peripheral to be connected with the smartphone
My Approach	Uses Machine learning with Random forest algorithm to provide high accuracy and speed and also is feasible with android and Linux based devices	

3. RESEARCH METHODOLOGY:

The proposed approach identifies rootkits that operate at the system's core layer using machine learning algorithms. Random Forest is a non-parametric machine learning method that is regarded as one of the most optimal solutions for this project since it makes decisions depending on the data it is provided with. It categorizes samples based on the vote count. The study focuses particularly on Android-level rootkits since the prevalence of these mobile devices and their vulnerabilities continues to rise. 1 provides a detailed description of why Android was selected for the project.

Setup & tools: In this project, Google colab pro was used to setup and build the whole model using Python code in the browser. Along with COLAB, many other machine learning python libraries like Scikit learn, seaborn, pandas, and numpy were also used.

Dataset: The dataset used in this research to train the model to detect the Android rootkit is taken from a public source. The dataset is taken from the research paper An Analysis of Android Malware Classification Services. The labels for this dataset are extracted from the Virus Total report of 2.47 million Android apk hashes. The dataset is made up of Sha256, Sha1, and md5 hashes. Since Sha1 hash is more trustworthy than md5, which can be changed, the dataset is even more trustworthy. The dataset also specifically classifies Android.rootkit and their hashes, which are then compared to the virus total and a hash checker to determine whether they are malicious

or benign. The main source of the dataset is AndroZoo. At the time of preparing the dataset, AndroZoo contained a little more than 13 million apps, collected from 14 different markets and repositories, including Google Play. **(*GitHub - mra12/labelingDataset*, no date; Rashed and Suarez-Tangil, 2021)**

Source	Samples	1st_seen Range
Malgenome	1.2 K	14 October 2009–12 June 2012
Contagio	1.6 K	25 February 2011–20 March 2018
Drebin	5.6 K	14 October 2009–10 August 2013
AMD	24 K	17 November 2010–14 May 2016
Palo Alto	104 K	7 November 2011–3 May 2019
OTX	116 K	26 September 2012–9 July 2020
VirusShare	170 K	11 April 2010–27 December 2019
AndroZoo	2.24 M	14 October 2009–3 November 2020
Total	**2.47 M**	**~late 2009 to late 2020**

Table1: List of various dataset that were used in the analysis of the current dataset.

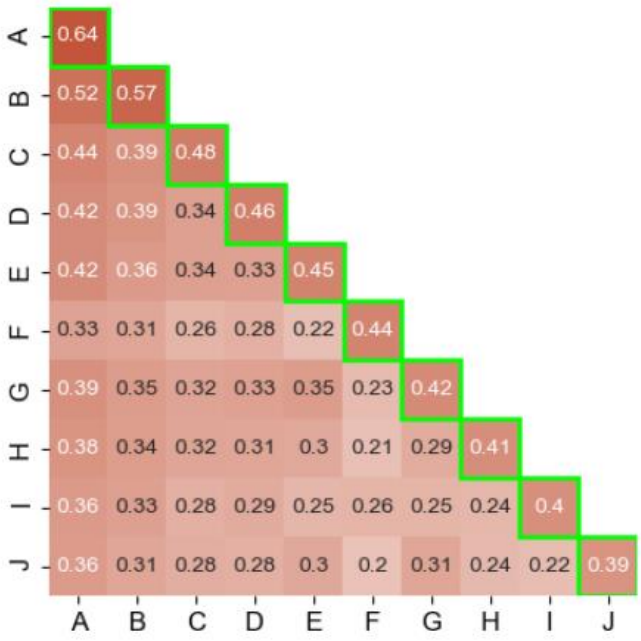


Table 2: Heatmap for the top 10 engines in coverage. (A): eset-nod32, (B): ikarus, (C): fortinet, (D): cat-quickheal, (E): nano-antivirus, (F): symantecmobileinsight, (G): avira, (H): cyren, (I): k7gw, (J):

F-secure. The cells with the light green edges are those that represent the single coverage for the engines

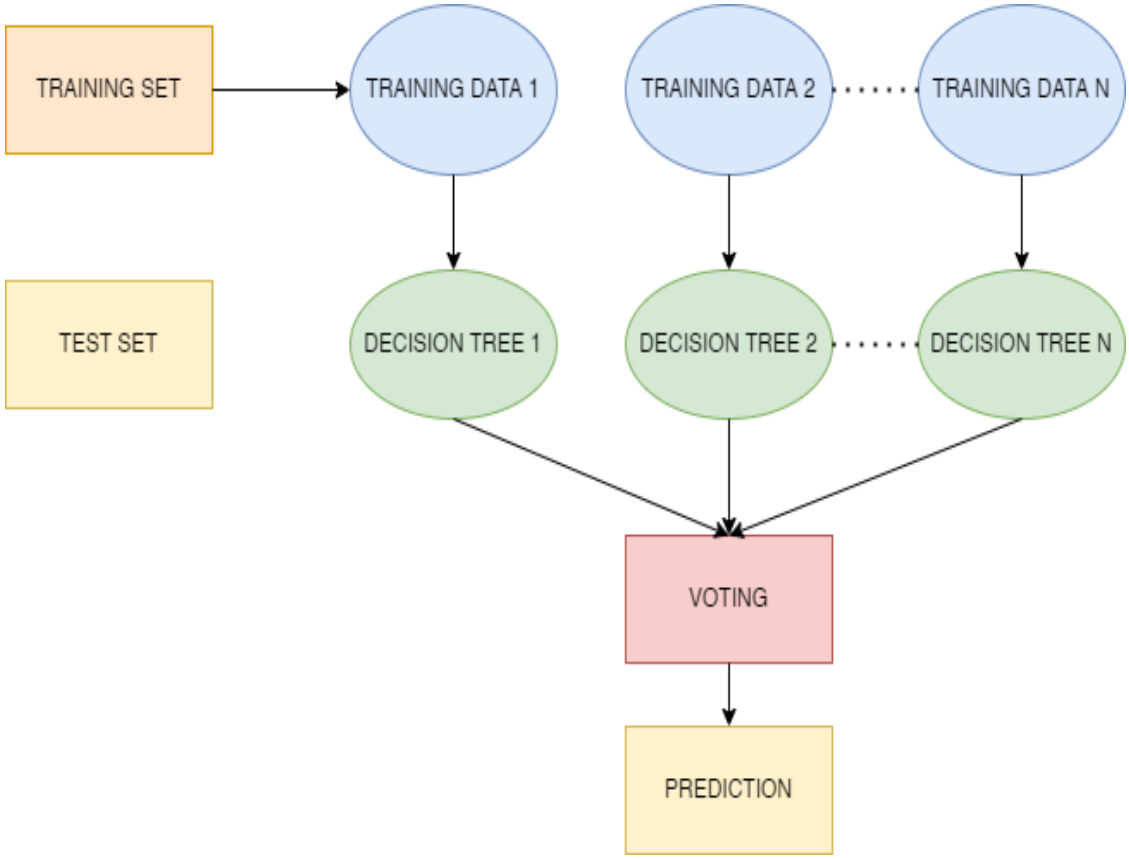
4. DESIGN SPECIFICATION:

Algorithm: Prior studies 6 on rootkits revealed both substantial benefits and drawbacks, but there was little or no research and design methodology that leveraged Random Forest as the primary machine learning approach for identifying an Android rootkit. Random forest is a supervised machine learning approach, which implies that supervised learning is the process of providing correct input and output data to the machine learning model. The goal of a supervised learning algorithm is to find a mapping function that moves the input variable (x) to the output variable (y). **(Sruthi, 2021)(*05.08-Random-Forests.ipynb - Colaboratory*, no date)**

Random forest is one of the powerful algorithms with many advantages,

- Due to the simplicity of basic decision trees, learning and prediction are very quick. Moreover, both operations are readily multithreaded since the individual trees are entirely self-contained entities.
- Multiple trees provide statistical classification: a probability assessment is produced by a majority choice among estimators.
- Due to the adaptability of the nonparametric model, it may perform well in circumstances where other methods fail.

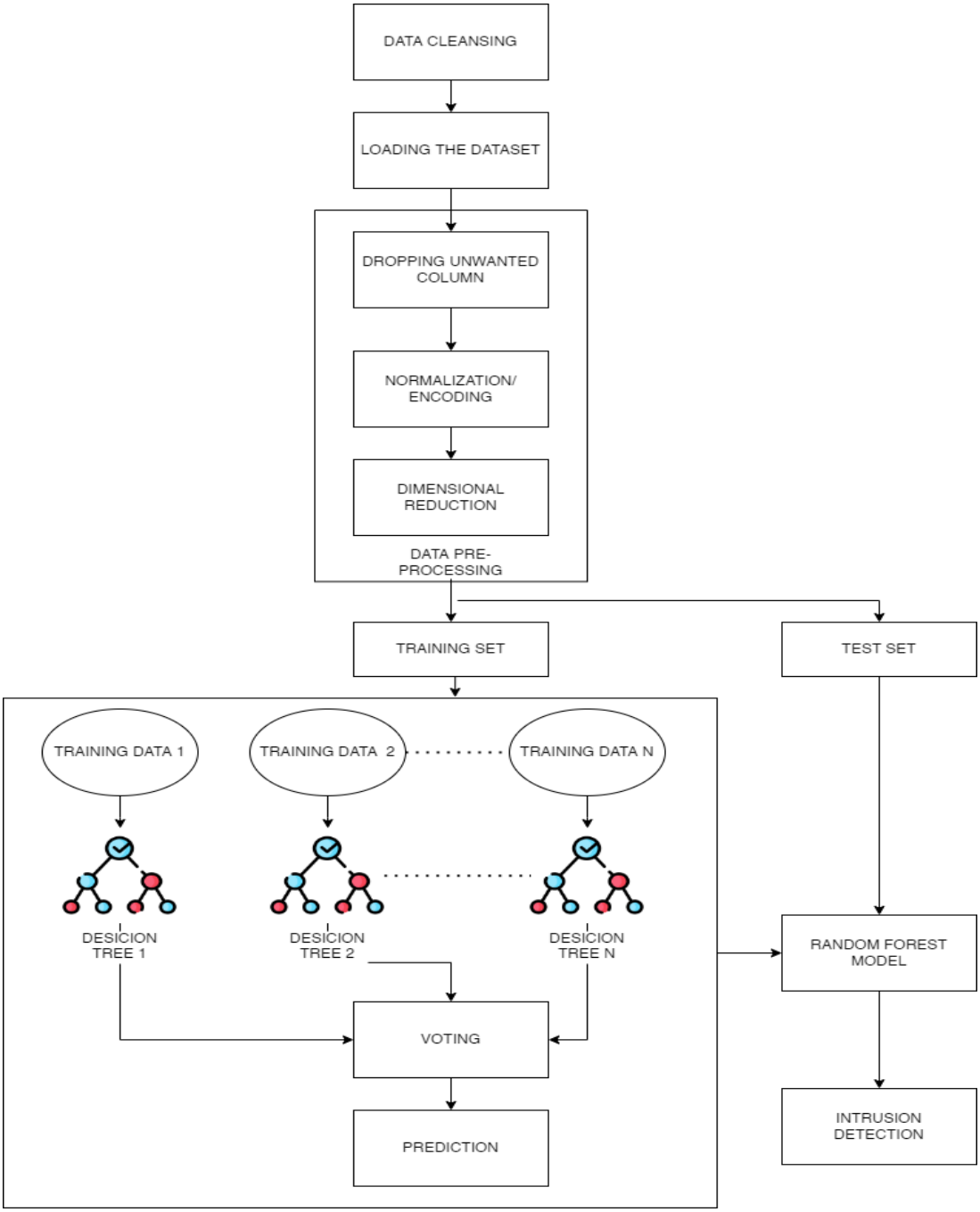
A random forest enhances the accuracy of the data set by calculating the mean of all the subsets using separate decision trees trained on the subsets. The end outcome is decided by the tree with the most votes. The algorithm's efficiency is proportional to the number of trees it employs. Below is an illustration of the random forest approach in practice.



The following are the steps to be taken throughout the designing of the model:

- First a suitable dataset was selected for the project.
- Data cleansing was performed with the selected dataset.
- All the crucial libraries for the project are imported.
- Dataset was imported into the Google colab platform.
- Checking for null values and dropping the unwanted columns.
- Encoding and normalization of the data.
- Performing ISOMAP on the columns for dimensional reduction.
- Splitting the dataset into training and test data.
- Applying GridSearch with Random forest classifier.
- Estimating and calculating accuracy, precision, recall, F1-score.
- Drawing a confusion matrix with Tp,Fn,Tn,Fp.

5. IMPLEMENTATION:



➢ DATA PRE-PROCESSING:

Here, the raw data is cleaned and formatted in a way that is appropriate to the ML preferred model. The dataset requires extensive cleansing through modification and normalization to

eliminate noise and improve precision. Steps such as data cleansing, changing the data type, and categorical data conversion are all part of the preprocessing phase.

Data cleaning, or the elimination of duplicate or unnecessary information from the original data set, improves the speed at which a machine learning model trains and the accuracy with which it performs its analysis. Finding and correcting any blanks or missing data in the dataset is an important initial step in fixing any issues that may have been discovered. One of the best ways to clean up data is to get rid of rows that have nothing in them. In this dataset the final labels were not classified, thus the final label classification was done manually by classifying the clean data as 0; Rootkits as 1 and other malwares as 2.

Convert Categorical Data to Numerical: Columns with null value are checked and those columns which are not necessary are dropped. Then, normalization and encoding operation where performed on the records which had string values. The majority of machine learning models need numerical representations of categorical data. However, some models are useful only with numerical information, whereas others are effective only with categorical features. Random forest can process both numerical and categorical data, although it is preferable to classify the data according to the problem statement in order to enhance the processing speed.**(*Significance of Data Transformation in Machine Learning* |, no date)**

ISOMAP, is an Unsupervised Machine Learning technique aimed at Dimensionality Reduction. As the dataset consist of 98 columns isomap was used for dimension reduction.

```
[ ]  from sklearn.manifold import Isomap

     mapping = Isomap(n_components =4)
     mapped_X = mapping.fit_transform(X)
```

DIVIDING THE DATASET INTO TWO GROUPS: TRAINING DATA AND TEST DATA:

The pre-processed dataset can be split into training and test dataset. The training and test data was split in ratio of 70:30 respectively.

➢ **TRAINING THE DATA SET:**

Using the programme scikit learn, the random forest algorithm was used in training set to get a more precise output. For this, the RandomForestClassifier class from the sklearn.ensemble package will be imported and used to fit the data. Before training the data, GridSearch algorithm was used to tuning the hyper parameters. auto, sqrt, log2, gini, entropy are the parameters of the random forest and GridSearch was used to select the best suitable parameter for the model.

```
[ ]  CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 3)
     CV_rfc.fit(x_train, y_train)

     GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=42),
                  param_grid={'criterion': ['gini', 'entropy'],
                              'max_depth': [4, 5, 6, 7, 8],
                              'max_features': ['auto', 'sqrt', 'log2'],
                              'n_estimators': [200, 500]})
```

The random forest method is based on the bagging principle, in which a distinct training subset is created from sample training data via replacement, and the final output is determined by a majority vote.

```
rfc1=RandomForestClassifier(random_state=42, max_features='auto', n_estimators= 500, max_depth=4, criterion='gini')
```

Step 1: 70% of the total 14242 records were chosen for training, and the results were evaluated.

Step 2: Independent decision trees are generated for each sample.

Step 3: An output is generated for each decision tree.

Step 4: For classification/ regression, the final result is taken into consideration based on Majority Voting or Averaging.

A final testing score of 97.12% was achieved.

6. EVALUATION:

Skleran.metrics library was ran to calculate few evaluation metrics like accuracy, precision, F1 score and recall

- **Accuracy:**

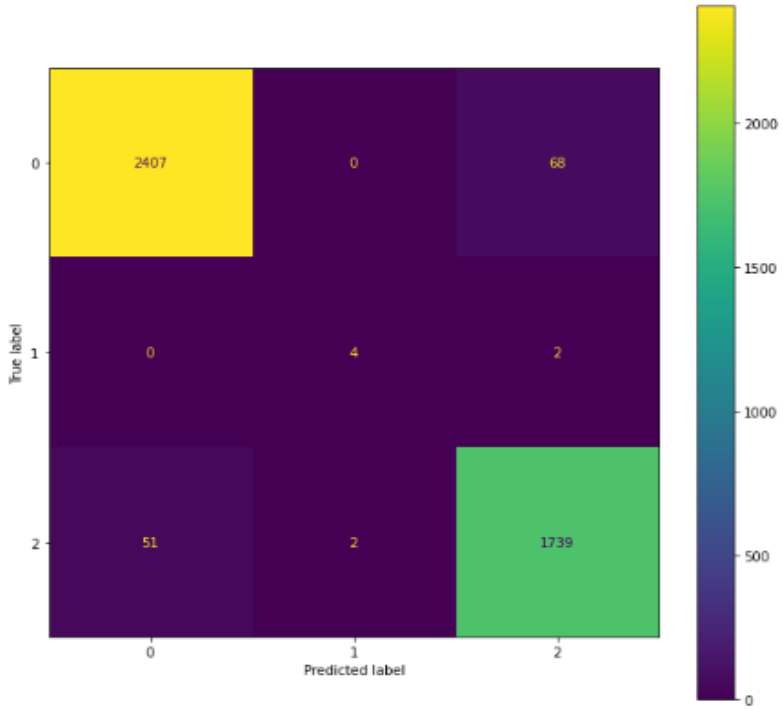
The model is evaluated on test data and it gives 97.12% accuracy.

```
[ ]  score = accuracy_score(y_test,pred)
     print("Testing Score: {:.2f} %".format(score*100))

     Testing Score: 97.12 %
```

- **Confusion matrix:**

In this paper there are three different prediction as there are three different final labels 0,1 and 3 which denotes clean, Rootkit and other malwares respectively. Thus the confusion matrix of this will be a 3*3 matrix.



From the figure, the value on the diagonal is the correctly predicted value of 0, 1&2. These values can be used to calculate the precision, recall and F1 score.

- **Precision**: In statistics, precision is defined as the number of accurately detected members of a class divided by the total number of times the model predicted that class to exist. Below is an example calculation of precision for label 0.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive} \qquad \text{Precision} = \frac{2407}{2458} = 0.98$$

Recall: The recall of a class is defined as the number of members of a class that were properly recognized by the classifier divided by the total number of members in that class. Below is an example calculation of recall for label 0.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative} \qquad \text{Recall} = \frac{2407}{2475} = 0.97$$

F1 score: The F1 score is less obvious since it combines precision and recall into a single statistic. F1 will be high if precision and recall are both high. If both are low, F1 will also be low. F1 will be low if one is high and the other is low. F1 is a simple approach to determine if a classifier is capable of accurately recognizing members of a class. Below is an example calculation of F1 score for label 0.

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall} \qquad \text{F1 score} = \frac{1.9012}{1.95} = 0.97$$

The figure below shows the scores calculated for rest of the labels 0, 1&2.

```
[ ]  print(classification_report(y_test, pred))
```

	precision	recall	f1-score	support
0	0.98	0.97	0.98	2475
1	0.67	0.67	0.67	6
2	0.96	0.97	0.97	1792
accuracy			0.97	4273
macro avg	0.87	0.87	0.87	4273
weighted avg	0.97	0.97	0.97	4273

6.5 DISCUSSION:

As was mentioned earlier, the rootkit operates as a module in Android's Linux kernel. As a result, it has the highest level of privilege on the Android phone and can be a very powerful tool for attackers. As a result, the rootkit based on the Android system is the primary focus of this paper because of the potential threat it poses to Android devices. Rootkits and other forms of malware, in addition to the model itself, were successfully identified after its implementation. Additionally, the model closed a gap in the existing research literature by performing detection based on Android systems using an innovative method for identifying rootkits using real-time hashes that were included in the dataset. If only there were an efficient amount of time and resources to spend on the study, then there are a number of different ways in which the effectiveness of this approach may be improved. In the next part of the future work, the concept of how the model may be improved so that it functions more effectively will be discussed.

7. CONCLUSION AND FUTURE WORK:

As previously stated, Smartphone-targeted cyber attacks are expanding daily, and the vast majority of ordinary users, who have little awareness of cyber attacks, are very susceptible to these sorts of assaults. Rootkits are very harmful since they conceal their existence. Therefore, it is crucial to do research specifically on Android smartphones.

The primary reason for using machine learning as a rootkit detection vector is so that it may be automated and learn from the datasets given. The random forest is renowned for its diversity. It may process data without sufficient cleansing and deliver very accurate results.

The purpose of the final report was to develop a machine learning algorithm-based rootkit detection method for Android smartphones. Therefore, it was accomplished with a great degree of precision and Android.Rootkits and other malware were also identified (they are classified by their labels).

In the future work, there is a great deal of research to be undertaken, which demands a great deal of time and resources. One of the approaches I would like to see included into future work is the

creation of a Sandbox environment and the execution of the APK files of different Android-specific applications. This allows for the identification of the harmful program's signature and the removal of the malware from the user's environment, hence increasing security.

8. VIDEO PRESENTATION LINK:

Final presentation.mp4

9. REFERENCES:

• *Distribution of Android malware 2019 | Statista* (no date). Available at: https://www.statista.com/statistics/681006/share-of-android-types-of-malware/ (Accessed: 12 August 2022).

• *Smartphone subscriptions worldwide 2027 | Statista* (no date). Available at: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ (Accessed: 12 August 2022).

05.08-Random-Forests.ipynb - Colaboratory (no date). Available at: https://colab.research.google.com/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.08-Random-Forests.ipynb#scrollTo=lnONOS7EzOiU (Accessed: 12 August 2022).

Bickford, J. *et al.* (2010) 'Rootkits on smart phones: Attacks and implications', *Workshop on Mobile Computing Systems and Applications*, pp. 49–54. Available at: http://www.cs.rutgers.edu/~vinodg/papers/technical_reports/tr654/tr654.pdf.

Bickford, J. E., Ganapathy, V. and Iftode, L. (2012) *ROOTKITS ON SMART PHONES: ATTACKS, IMPLICATIONS, AND ENERGY-AWARE DEFENSE TECHNIQUES*.

Bojan Jovanović (2021) *A Not-So-Common Cold: Malware Statistics in 2021 | DataProt*. Available at: https://dataprot.net/statistics/malware-statistics/ (Accessed: 12 August 2022).

Brodbeck, R. C. (2012) 'Covert Android Rootkit Detection: Evaluating Linux Kernel Level Rootkits on the Android Operating System', p. 98. Available at: http://www.dtic.mil/dtic/tr/fulltext/u2/a563041.pdf%5Cnhttp://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA563041.

GitHub - mra12/labelingDataset (no date). Available at:

https://github.com/mra12/labelingDataset (Accessed: 12 August 2022).

Guri, M. *et al.* (2015) 'JoKER: Trusted detection of kernel rootkits in android devices via JTAG interface', *Proceedings - 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2015*. IEEE, 1, pp. 65–73. doi: 10.1109/Trustcom.2015.358.

javatpoint (2019) 'Supervised Machine learning - Javatpoint'. Available at: https://www.javatpoint.com/supervised-machine-learning (Accessed: 12 August 2022).

Jiang, X., Lora, M. and Chattopadhyay, S. (2020) 'Efficient and trusted detection of rootkit in IoT devices via offline profiling and online monitoring', *Proceedings of the ACM Great Lakes Symposium on VLSI, GLSVLSI*, pp. 433–438. doi: 10.1145/3386263.3406939.

Kuzminykh, I. and Yevdokymenko, M. (2019) 'Analysis of Security of Rootkit Detection Methods', in *2019 IEEE International Conference on Advanced Trends in Information Theory, ATIT 2019 - Proceedings*. Institute of Electrical and Electronics Engineers Inc., pp. 196–199. doi: 10.1109/ATIT49449.2019.9030428.

Lab, L. T. (2021) *Rooting Malware Makes a Comeback: Lookout Discovers Global Campaign*. Available at: https://www.lookout.com/blog/lookout-discovers-global-rooting-malware-campaign (Accessed: 12 August 2022).

Luckett, P., Todd McDonald, J. and Dawson, J. (2016) 'Neural Network Analysis of System Call Timing for Rootkit Detection', *Proceedings - 2016 Cybersecurity Symposium, CYBERSEC 2016*. IEEE, pp. 1–6. doi: 10.1109/CYBERSEC.2016.008.

Nadim, M., Lee, W. and Akopian, D. (2021) 'Characteristic features of the kernel-level rootkit for learningbased detection model training', *IS and T International Symposium on Electronic Imaging Science and Technology*, 2021(3), pp. 1–7. doi: 10.2352/ISSN.2470-1173.2021.3.MOBMU-034.

Nagy, R. *et al.* (2021) 'Rootkit Detection on Embedded IoT Devices', *Acta Cybernetica*. University of Szeged, Institute of Informatics, 25(2), pp. 369–400. doi: 10.14232/ACTACYB.288834.

Ranajit Singh Mehroke (2019) *'Attacks on the Android Platform' by Ranajit Singh Mehroke*. Available at: https://repository.stcloudstate.edu/msia_etds/82/ (Accessed: 12 August 2022).

Rashed, M. and Suarez-Tangil, G. (2021) ‘An analysis of android malware classification services’, *Sensors*. MDPI AG, 21(16). doi: 10.3390/S21165671.

Sayadi, H. *et al.* (2021) ‘Towards accurate run-time hardware-assisted stealthy malware detection: A lightweight, yet effective time series cnn-based approach†’, *Cryptography*. MDPI, 5(4). doi: 10.3390/cryptography5040028.

Significance of Data Transformation in Machine Learning | (no date). Available at: https://www.analyticsinsight.net/significance-of-data-transformation-in-machine-learning/ (Accessed: 12 August 2022).

Singh, B. *et al.* (2017) ‘On the detection of Kernel-level rootkits using hardware performance counters’, in *ASIA CCS 2017 - Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security*. Association for Computing Machinery, Inc, pp. 483–493. doi: 10.1145/3052973.3052999.

Sruthi, E. R. (2021) *Random Forest | Introduction to Random Forest Algorithm, AnalyticsVidya.com*. Available at: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/ (Accessed: 12 August 2022).

Tian, D. *et al.* (2019) ‘A Kernel Rootkit Detection Approach Based on Virtualization and Machine Learning’, *IEEE Access*. Institute of Electrical and Electronics Engineers Inc., 7, pp. 91657–91666. doi: 10.1109/ACCESS.2019.2928060.

Wang, Z. *et al.* (2008) ‘Countering persistent kernel rootkits through systematic hook discovery’, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 21–38. doi: 10.1007/978-3-540-87403-4_2.

Xie, X. and Wang, W. (2013) ‘Rootkit detection on virtual machines through deep information extraction at hypervisor-level’, *2013 IEEE Conference on Communications and Network Security, CNS 2013*. IEEE, pp. 498–503. doi: 10.1109/CNS.2013.6682767.