ISSN 2278-3091
Volume 9, No.4, July – August 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse14942020.pdf
https://doi.org/10.30534/ijatcse/2020/14942020

A Machine Learning approach for analyzing and detecting Android malicious applications

V. Joseph Raymond [1], **R.Jeberson Retna Raj** [2]
[1] Sathyabama Institute of Science and Technology, India, josephrv@srmist.edu.in
[2] Sathyabama Institute of Science and Technology, India, jebersonretnaraj.it@sathyabama.ac.in

ABSTRACT

Android applications are available in the google cloud apps market. Starting from the normal functionalities like calling, messaging, and camera to advanced functionalities like online banking, online shopping, there is no limit to how we can make use of mobile phones. Just like there is no limit to functionalities of the mobile phone, there is no limit to the amount of information available on mobile phones. When you say information, it is confidential information including personal information, username, and password and card details. There are already many cases reported about information leakage by compromising a mobile phone. The important point to be noticed here is that the medium through which mobile phones are providing these functionalities, and that medium is called an application. There are millions of applications in the Play Store and App Store which come with different functionalities. The only way to stop this problem is by stopping the user from downloading malicious applications from the Play Store. But the main challenge in this solution leads to a question, which is "how the user will distinguish between a malicious application and benign application". There are millions of applications in play Store and considering play Store is a trusted media, it is very unlikely to raise any suspicion over an application to a normal user. We propose a novel approach to perform static and dynamic analysis of malicious payload and compare it with the genuine application.

Key words: Google Cloud App, Malicious Payload, Android apk files, Classifiers

1. INTRODUCTION

One side, the defenders, who will always try to stop the attackers by using updated technologies to secure their system and thereby securing the valuable information in their system from cloud apps. It is time we move from this existing solution as it can no longer prevent the attackers from getting hands-on our information. Without programming the machine can learn can be obtained, the outcome can be developed for android applications [1].In all ways, it will be more efficient and right than all the existing solutions. There are three kinds of machine learning algorithms:

1.1. Supervised Learning

This is labeled supervision where the data is aligned with the appropriate answer. After this, we collect new trained data from the algorithm and then match the outcome from the data.

1.2 Unsupervised Learning

This is not a labeled approach, where the algorithm will work without any help. The machine will find unsorted data from the pattern without having any data.

1.3 Reinforcement Learning

This approach gains maximum points based upon some situations [6-8] Henceforth this approach similar to the evaluation process with the answer key where predication is based on experience from various levels. The agent will get rewards from many obstacles between him succeeding in his levels as shown in Figure 1.

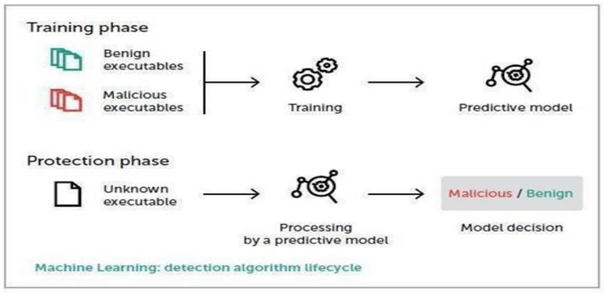


Figure 1: Understanding the training phase

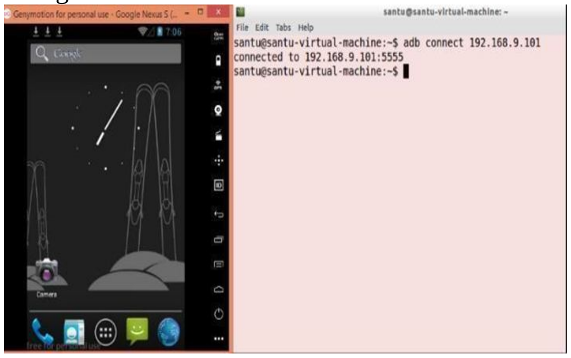
2. APP ANALYSIS

2.1 App Compatibility

This is known as the "app compatibility". Android applications are capable of running on various devices such as mobile phones, tablets, television and so on. As of now, the scope of this application only allows it to run mobile phones.

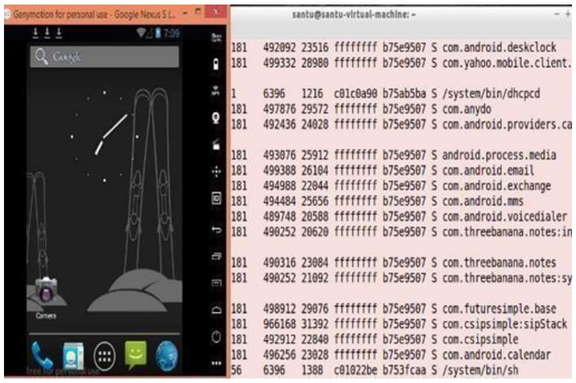
2.2 Static Analysis of Apps

Malware analysis involves both static and dynamic analysis. Static analysis is one of the malware analyzing technique which will look over one of the important files in the .apk file containing all the permissions required by a particular application for its successful run from the manifest file. This Use either SI (MKS) or CGS as primary units. (SI units are strongly encouraged.) English units may be used as secondary is what we have used for static analysis. [2][22]Static analysis will check all the permissions of the manifest file to detect malicious actions. These permissions are the input to the machine learning algorithm. The process is divided into three main steps as given below. We can extract useful information from the device such as serial number, device name, access specifier and encryption status [9-11]. This process checks whether the device is rooted or not henceforth gaining some privileges. Stock recovery mode can be used as an alternative. Searching evidence from web browsers that have established a connection with the device. Computation of hash using the md5deep tool and find its digital signature. We will connect the device with the platform as shown in Figure 2. The installation of a malicious App can be done through Adb install command as shown in Figure 3. Adb Logcat is used to track and monitor the presence of malicious activity as shown in Figure 4. The Adb Pull operation will take the file permissions and copy the apk file in the destination as shown in Figure 5.



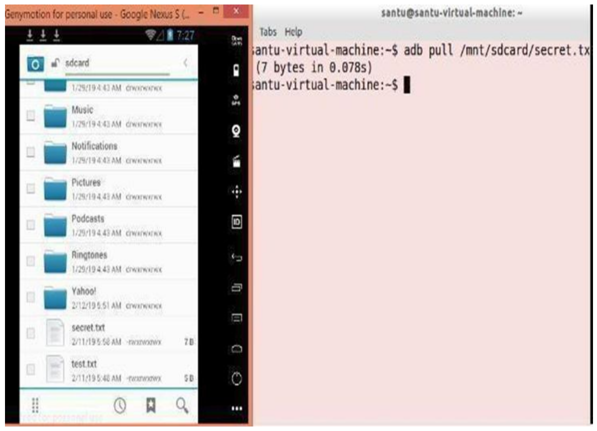
```
santu@santu-virtual-machine:~$ adb connect 192.168.9.101
connected to 192.168.9.101:5555
santu@santu-virtual-machine:~$
```

Figure 2: ADB Connect



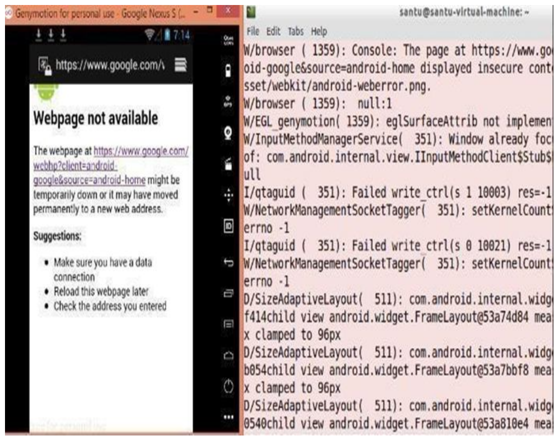
```
181   492092 23516 ffffffff b75e9507 S com.android.deskclock
181   499332 28980 ffffffff b75e9507 S com.yahoo.mobile.client.

1     6396   1216  c01c0a90 b75ab5ba S /system/bin/dhcpcd
181   497876 29572 ffffffff b75e9507 S com.anydo
181   492436 24028 ffffffff b75e9507 S com.android.providers.ca

181   493076 25912 ffffffff b75e9507 S android.process.media
181   499388 26104 ffffffff b75e9507 S com.android.email
181   494988 22044 ffffffff b75e9507 S com.android.exchange
181   494484 25656 ffffffff b75e9507 S com.android.mms
181   489748 20588 ffffffff b75e9507 S com.android.voicedialer
181   490252 20620 ffffffff b75e9507 S com.threebanana.notes:in

181   490316 23084 ffffffff b75e9507 S com.threebanana.notes
181   490252 21092 ffffffff b75e9507 S com.threebanana.notes:sy

181   498912 29076 ffffffff b75e9507 S com.futuresimple.base
181   966168 31392 ffffffff b75e9507 S com.csipsimple:sipStack
181   492912 22840 ffffffff b75e9507 S com.csipsimple
181   496256 23028 ffffffff b75e9507 S com.android.calendar
56    6396   1388  c01022be b753fcaa S /system/bin/sh
```

Figure 3: ADB Install



```
santu-virtual-machine:~$ adb pull /mnt/sdcard/secret.tx
(7 bytes in 0.078s)
santu-virtual-machine:~$
```

Figure 4: ADB Logcat



```
W/browser ( 1359): Console: The page at https://www.go
oid-google&source=android-home displayed insecure cont
sset/webkit/android-weberror.png.
W/browser ( 1359):  null:1
W/EGL_genymotion( 1359): eglSurfaceAttrib not implemen
W/InputMethodManagerService(  351): Window already foc
of: com.android.internal.view.IInputMethodClient$Stub$
ull
I/qtaguid (  351): Failed write_ctrl(s 1 10003) res=-1
W/NetworkManagementSocketTagger(  351): setKernelCount
errno -1
I/qtaguid (  351): Failed write_ctrl(s 0 10021) res=-1
W/NetworkManagementSocketTagger(  351): setKernelCount
errno -1
D/SizeAdaptiveLayout(  511): com.android.internal.widg
f414child view android.widget.FrameLayout@53a74d84 mea
x clamped to 96px
D/SizeAdaptiveLayout(  511): com.android.internal.widg
b054child view android.widget.FrameLayout@53a7bbf8 mea
x clamped to 96px
D/SizeAdaptiveLayout(  511): com.android.internal.widg
0540child view android.widget.FrameLayout@53a810e4 mea
```

Webpage not available

The webpage at https://www.google.com/webhp?client=android-google&source=android-home might be temporarily down or it may have moved permanently to a new web address.

Suggestions:

- Make sure you have a data connection
- Reload this webpage later
- Check the address you entered

Figure 5: ADB Pull

2.3 Root of Trust

This approach is an understanding of manifest file where we find important details about commands, system files, and folders where the user finds full information about the system. This provides information to machine learning algorithms that provide the need to act on android devices [12-14].In turn, achieving the effective gaining of system credential data that can even break the integrity of the system. These contain a set of functions that computing models will consider. This is always suited for embedded applications. The functionalities behind these are encryption, unauthorized access, rootkit detection, improper read and write operations and issues with digital rights. It is a harder approach to review such mechanisms. The overall proposed system architecture: In static analysis, the downloaded app can be analyzed and examine the application behavior using dynamic analysis.

2.4 Apps: Dynamic analysis

Another way of doing malware analysis is known as dynamic analysis. This approach is the behavioral analysis where the approach is purely based on run time. [4]The main focus on features of the event, networking behaviors, battery issues, log-based analysis, gaining privileges from root devices.

2.5 Combining Static and Dynamic Analysis

Our proposed system implements both static and dynamic approaches where only one approach doesn't yield those effective results. [5]Out detection mechanism will work based on doing static analysis before installing the payload and then dynamic analysis execution in a safe environment. The malicious apk files are installed in a sandboxing place where it's an effective approach. The combination result always provides effective output.

2.6 Code Obfuscation& Utility SDK

This technique makes it very difficult for victims to understand the nature of the impact. This includes cheating of manifest file, linking the payload during execution by the processor. The difference between the benign app and the malicious app is not that easy to classify, we take unknown samples and apply machine learning techniques and suggest a decision support system. [4][21]The extraction of the system call is a very important feature under this approach. Android Monkey is a utility SDK tool that simulates the nature of human thinking comparing with applications. A log-based vulnerability assessment is performed. We have to mainly focus on removing the noise and achieving optimal featured vector that can be used to validate the robustness of various adversarial attacks. With this approach, we achieve the ranked system call resulting in identifying malicious applications.

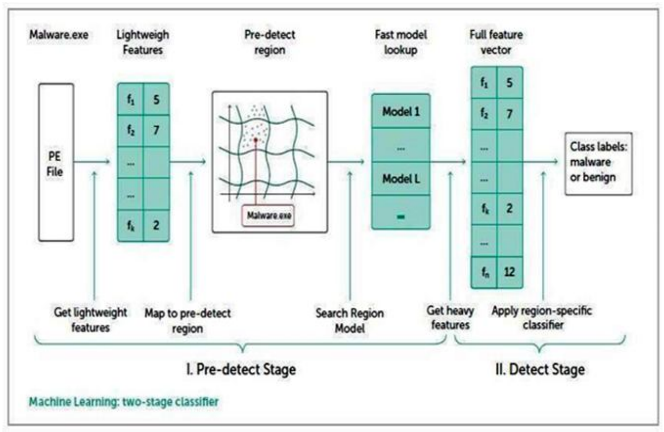


Figure 6: Proposed Architecture

3.UNDERSTANDING OF PROBLEM STATEMENT

There are so many vulnerable zero-day exploits that don't have signatures available in modern antivirus scanners. The expertise malware creators can easily bypass signature by code obfuscation as discussed earlier. [15-17]The success rate of finding such entities' in the real world is still a changeling scenario. Although there is enough security checking while uploading the files to the google play store but can't give a complete solution.

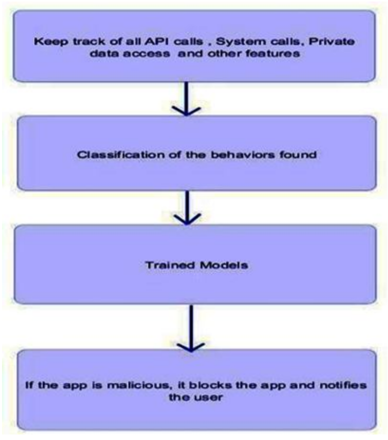

Keep track of all API calls , System calls, Private data access and other features
Classification of the behaviors found
Trained Models
If the app is malicious, it blocks the app and notifies the user

Figure 7: Stages of Implementation

In our proposal, we suggest a machine learning approach that can filter applications that can create a harmful impact on the system where it is installed as shown in Figure 6. Instead of taking the signature, we create trained data set from samples that give a better solution. The success rate obtained through this classifier approach gives better outcomes, thus contributes to the current industrial revolution. We take some training data from online resources like kudos train models on that data and predict with the new data. The model makes a prediction and helps correct code with errors in prediction. The detection happens in two stages. The PE header file from the malicious payload taken as input. [18-19]Light features are extracted from static, apply pre detect region and then region-specific classifier in the detection phase. This approach gives an effective approach for classifying malicious apps from genuine app as shown in Figure7.

4.RESULTS

The dataset used here is the Android Malware Dataset taken from gut hub account. It consists of several malicious predictor variables and one target variable. The various medical variables are call records, gps tracking and SMS etc. It contains 768 rows and 9 columns. The dataset file is in a .csv (Comma Separated Values) format. Using the help of Python’s inbuilt library Pandas, which is a data frame library, we import the file into our Python environment [21]. The other libraries that are imported into the environment are:

4.1 Numpy

A library that is used to mainly operate with large dimensional arrays and matrices, providing high-level mathematical functionalities to work on data.

4.2 Matplotlib

The library that provides Python with the functionality of plotting graphs and plots. It works in tandem with NumPy. Pandas have a function named read_csv(), which essentially reads a file of the format (.csv).

Once the dataset is loaded into the environment, we can check the dimensions of the dataset by the function .shape () which returns the number of rows and columns. The basic lookup of the data is done, by using the inbuilt commands .head () and .tail () which print the number of rows from start to finish of the dataset. After acquiring the dataset, we check whether we can roll out any improvements to the dataset. The data bear some of the operations like initializing the variables, refining the data, creating possible labels. In this case, the dataset consists of a parameter SMS sending, this section has a weak connection to the commitment of an individual being affected by the malware. Hence, the column can be deleted for the analysis. In this step, the calculation of numerical aspects of data, such as number of cases, average based on columns. The dataset consists of values for the persons who affected by malicious Apk's. After the calculation and count for each case, the result as shown below:

Apk's with Call Log's:268
Apk's with SMS Attack and GPS: 500

Based on data, over 35% of the person have been analyzed with the presence of malicious payload. One of the difficult steps in analysis are splitting the dataset for training and testing. This procedure is basically accomplish to check whether the test data and training data are different, because the model to be tested based on the training process.

Using Bayes' standard:

$$p(C_j \mid x_1, x_2, \ldots, x_d) \propto p(x_1, x_2, \ldots, x_d \mid C_j)\, p(C_j) \quad (1)$$

And revise the posterior as:

$$p(X \mid C_j) \propto \prod_{k=1}^{d} p(x_k \mid C_j)$$
$$p(C_j \mid X) \propto p(C_j) \prod_{k=1}^{d} p(x_k \mid C_j) \quad (2)$$

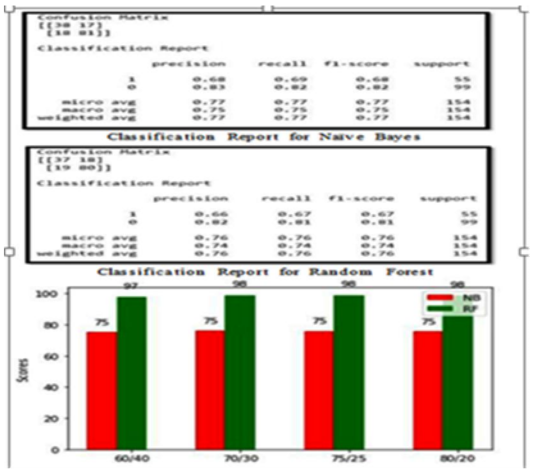

```
Confusion Matrix
[[38 17]
 [18 81]]

Classification Report

              precision    recall  f1-score   support

           1       0.68      0.69      0.68        55
           0       0.83      0.82      0.82        99

   micro avg       0.77      0.77      0.77       154
   macro avg       0.75      0.75      0.75       154
weighted avg       0.77      0.77      0.77       154
```
Classification Report for Naive Bayes
```
Confusion Matrix
[[37 18]
 [19 80]]

Classification Report

              precision    recall  f1-score   support

           1       0.66      0.67      0.67        55
           0       0.82      0.81      0.81        99

   micro avg       0.76      0.76      0.76       154
   macro avg       0.74      0.74      0.74       154
weighted avg       0.76      0.76      0.76       154
```
Classification Report for Random Forest

Figure 8: Result Comparison

By using above Bayes' standard, the variable X with a class level Cj that complete the most astonishing posterior probability. The indicator (autonomous) simplifies the ordering task automatically, because it allows the class restrictive densities p (xk | Cj) that can be determined for every factor respectively, i.e., it decrease a multidimensional activity to various one-dimensional ones. Essentially, Naive Bayes eliminates a high-dimensional density estimation undertaken by the one-dimensional part density estimation. Once we import the model into a variable, giving input to the training data, fitting the Gaussian Naïve Bayes model using .fit() function, then the performance of classification of array in each raining and testing variables can be done. This classification is the key operation for prediction of the input data according to the Naïve Bayes model. The accuracy of the model is then predicted by the comparison of predicted model with the original model. This can be done with the help of the metrics library present in sklearn. The simple test model graph shows the comparison active adversaries effect avoiding detection and the new version of malicious files as shown in Figure 8.

5.CONCLUSION AND FUTURE WORK

The malware writers can change signatures dynamic at run time, henceforth comparison done by the antivirus scanner's comparing the updated database pattern along the obtained signature will not have a match and bypass the regular check routine.[20] It takes time for identifying behavior or the nature of the payload. The machine learning algorithm henceforth provides an effective measure to counterattack these issues. This approach can be further extended with a deep learning approach using CNN or RNN based approach providing some detailed subset of training data. They provide complete protection satisfying the basic needs of the victims from spoofing and online attacks. An app shared via google play may not be modified or replaced other than the creator. Likewise, an app cannot download any executable files like dex or jar file This restrict does not practice to code that runs in a virtual system and get entry to Android APIs (together with JavaScript in a web view or browser). In this study, the survey of static and dynamic analysis of malware detection for Android application using machine learning.

REFERENCES

1. Martín, Ignacio, José Alberto Hernández, and Sergio de los Santos. "**Machine-learning based analysis and classification of android malware signatures**." *Future Generation Computer Systems* 97 (2019): 295-305. https://doi.org/10.1016/j.future.2019.03.006
2. Kabakus, Abdullah Talha, and Ibrahim Alper Dogru. "**An in-depth analysis of Android malware using hybrid techniques**." *Digital Investigation* 24 (2018): 25-33. https://doi.org/10.1016/j.diin.2018.01.001

3. Nguyen-Vu, Long, Jinung Ahn, and Souhwan Jung. "**Android fragmentation in malware detection.**" *Computers & Security* 87 (2019): 101573.

4. Zhang, Li, Vrizlynn LL Thing, and Yao Cheng. "**A scalable and extensible framework for android malware detection and family attribution**." *Computers & Security* 80 (2019): 120-133.
5. Amin, Muhammad, et al. "**Static malware detection and attribution in android byte-code through an end-to-end deep system**." *Future Generation Computer Systems* 102 (2020): 112-126. https://doi.org/10.1016/j.future.2019.07.070
6. Yen, Yao-Saint, and Hung-Min Sun. "**An android mutation malware detection based on deep learning using visualization of importance from codes**." *Microelectronics Reliability* 93 (2019): 109-114.
7. Rehman, Zahoor-Ur, et al. "**Machine learning-assisted signature and heuristic-based detection of malwares in Android devices**." *Computers & Electrical Engineering* 69 (2018): 828-841.
8. Palumbo, Paolo, et al. "**A pragmatic android malware detection procedure**." *Computers & Security* 70 (2017): 689-701. https://doi.org/10.1016/j.cose.2017.07.013
9. Calleja, Alejandro, et al. "**Picking on the family: Disrupting android malware triage by forcing misclassification**." *Expert Systems with Applications* 95 (2018): 113-126.
10. Hu, Xinwen, Yi Zhuang, and Fuyuan Zhang. "**A security modeling and verification method of embedded software based on Z and MARTE.**" *Computers & Security* 88 (2020): 101615.
11. Wang, Shanshan, et al. "**A mobile malware detection method using behavior features in network traffic.**" *Journal of Network and Computer Applications* 133 (2019): 15-25.
12. Zimmermann, Verena, and Nina Gerber. "**The password is dead, long live the password–A laboratory study on user perceptions of authentication schemes.**" *International Journal of Human-Computer Studies* 133 (2020): 26-44. https://doi.org/10.1016/j.ijhcs.2019.08.006
13. Garg, Shivi, and Niyati Baliyan. "**A novel parallel classifier scheme for vulnerability detection in android**." *Computers & Electrical Engineering* 77 (2019): 12-26.
14. Vinod, P., Akka Zemmari, and Mauro Conti. "**A machine learning based approach to detect malicious android apps using discriminant system calls.**" *Future Generation Computer Systems* 94 (2019): 333-350.
15. Garg, Shivi, and Niyati Baliyan. "**Data on vulnerability detection in android.**" *Data in brief* 22 (2019): 1081-1087.
16. Han, Weijie, et al. "**MalInsight: A systematic profiling based malware detection framework**." *Journal of Network and Computer Applications* 125 (2019): 236-250.
17. Kumar, Rakesh, and Rinkaj Goyal. "**On cloud security requirements, threats, vulnerabilities and countermeasures: A survey**." *Computer Science Review* 33 (2019): 1-48. https://doi.org/10.1016/j.cosrev.2019.05.002
18. Singh, Ashish, and Kakali Chatterjee. "**Cloud security issues and challenges: A survey.**" *Journal of Network and Computer Applications* 79 (2017): 88-115.
19. Alzaylaee, Mohammed K., Suleiman Y. Yerima, and Sakir Sezer. "**DL-Droid: Deep learning based android malware detection using real devices**." *Computers & Security* 89 (2020): 101663.
20. G. Geetha, Dr.K.Mohana Prasad,"**Prediction of Diabetics using Machine Learning**", International Journal of Recent Technology and Engineering, (IJRTE) ISSN: 2277-3878, Volume-8 Issue-5, January 2020 https://doi.org/10.35940/ijrte.E6290.018520
21. Alqurashi, Reem K., et al. "**Cyber Attacks and Impacts: A Case Study in Saudi Arabia.**" *International Journal of Advanced Trends in Computer Science and Engineering* 9.1 (2020). https://doi.org/10.30534/ijatcse/2020/33912020
22. Gadade, H. D., and D. K. and Kirange. "**Machine Learning Approach towards Tomato Leaf Disease Classification.**" *International Journal of Advanced Trends in Computer Science and Engineering* 9.1 (2020): 490-495. https://doi.org/10.30534/ijatcse/2020/67912020