FUDMA Journal of Sciences (FJS)
ISSN online: 2616-1370
ISSN print: 2645 - 2944
Vol. 3 No. 4, December, 2019, pp 266 –275

BEHAVIOR-BASED DETECTION: AN APPROACH FOR SECURING ANDROID SYSTEMS AGAINST ZERO-DAY MALWARE ATTACKS

[1]Joshua Abah, [2]Adati Elkanah Chahari, [3]Esther Alu Samuel, [4]Emmanuel P. Musa

[1]Department of Computer Engineering, University of Maiduguri, Maiduguri
[2]Department of General Studies Education, School of General Education, Federal College of Education, Yola
[3]Department of Computer Science, Nasarawa State University, Shabu-Lafia Campus, Lafia
[4]Department of Computer Science, Ramat Polytechnic Maiduguri

Corresponding author's Emails: abah@unimaid.edu.ng

ABSTRACT
We present behaviour-based detection as an approach to mitigating zero-day attacks on Android. This is as a result of the drawbacks of signature-based approach commonly in use in most antivirus engines. The Signature-based approach requires the analysis and storage of signature strings of malware with which new attacks are compared. This makes the detection of new attacks whose signatures have not been gotten impossible. For these attacks to be detected, patches must be developed for them. This unknown attack is referred to as zero-day attacks. Moreover, developing patches takes time creating a vulnerability window that could be exploited hence, there is the need to be able to detect zero-day attacks in real-time. To demonstrate the capability of detecting zero-day attacks, dynamic analysis of applications was adopted in this research. A detection system was developed for the Android system and features were extracted from the device and used to analyze the behaviour of the system. The K-Nearest Neighbour (KNN) classifier was used and results showed that this approach has 93.75% accuracy and 6.25% error rate. The Area Under Curve (AUC) of the Receiver Operating Characteristics (ROC) stands at 0.996 out of 1. This result showed that behavioural detection promises a future for malware detection with respect to zero-day detection. It is recommended that the features be extended to include features at a lower level of granularity that represents system-wide behaviour. In addition, this approach should be adopted by other mobile platforms besides Android.

Keywords: Android, Attacks, Behaviour-based detection, Exploit, Malware, Signature-based detection, Smartphones, Vulnerability, Zero-Day.

INTRODUCTION

Mobile devices have drastically become a ubiquitous computing and storage platform with increasing capacity, complexity, and usage.Among these devices, Android holds a large percentage of the market share with over 220 million mobile devices running Android this figure corresponds to well over 78% of Smartphones sold to consumers worldwide (Statista, 2015). Based on unit shipments of these smart devices, Android holds the highest percentage of global Smartphone Operating Systems (OSs).

The availability of Smartphones at relatively low prices has led to an accelerated migration of feature phone users to Smartphones; tasks previously carried out on laptops and PCs are now migrated to Smartphones making the smartphone OS market to experience fast growth in most emerging countries, including Nigeria, India, Russia and Mexico (Gartner, 2015). This trend continued to benefit Android, which saw its market share grow by 2.2 percentage points in 2014, and 32 percent year on year. Making Android the most used Smartphone's Operating system in the World (Gartner, 2015). Android is open source with a huge user community and documentation; it allows any programmer to develop and publish Applications to both the Official or Unofficial market (Srikanth, 2012). It has a very huge adoption and market penetration globally. Android was predicted to be the most used mobile Smartphone platform by 2014 (You *et al*., 2014) which has become a reality.

The ubiquity of the Android platform and indeed the Smartphones, in general, has not gone unnoticed by malware developers. Rather, this ubiquitous gain of Android carries along with it some security risks in terms of malware attacks targeted at this platform. Mobile devices and Android, in particular, has become a target of attacks. Android's popularity came with a cost as it has become a target for attacks for most malware developers. Although other mobile platforms suffer the same fate, the Android platform is the worst hit (Denis, 2012).

There are already well-known and documented cases of Android malware in both official and unofficial markets (Yajin*et al*., 2012). With known malware nefarious capabilities and effects, the detection of malware is an area of major concern not only to the research community but also to the general public. Malware attack is a challenging issue among the Android user community. It, therefore, becomes necessary to make the platform safe for users by providing defense mechanism especially against malware (Joshua *et al*., 2015). Techniques that researchers develop for malware detections are realized through the implementation of malware detectors (Nwokedi & Aditya, 2007). Malware detectors are the primary tools in defense against malware and the quality of such detectors is

determined by the techniques they employed. Intrusion detection methods can be classified as host-based, cloud-based or social collaboration (Srikanth, 2012) and the technique adopted for each of these methods could be signature-based, anomaly-based or virtual machine-based.

Most detection and antivirus engines in use today in the fight against malware intrusion adopt the signature-based approach in the prevention and detection of malware. While this technique could be effective for known malware; malware whose signatures have been obtained, analyzed and stored, it is practically impossible for signature-based intrusion detection engines to detect zero-day attacks; attacks that are not known. This makes zero-day attacks; malicious attacks that identify a vulnerability and exploits it before it becomes known to the software vendor and the end-users a difficult strain of malware to deal with as most antivirus software available today are based on signatures which cannot detect zero-day attacks. A malicious attack can use the exploit to downloadmalware, spyware, adware, phishing software, or any other kind of malicious code with criminal intent (Spamlaws, 2017).

Similarly, for signature-based detection engines to be able to detect new attacks, update patches must be developed. Zero-day threats are released and propagated into the wild before security vendors can issue protection against them. Malware can attack by targeting vulnerabilities in Operating systems and Applications. In the advent of the discovery of a weakness in commercial Applications, the vendor will have to write a patch to secure the software against attacks. The problem with this solution is that it takes time to develop or write patches. According to Oberheide*et al*., (2008), it takes approximately 45 days to successfully develop a patch and so systems or devices can be compromised before the vulnerability is fixed. Unpatched programs on your system increase your risk of a successful attack by a zero-day threat.

The limitations of a signature-based approach to intrusion detection show a major drawback in the capability of signature-based detection engines to effectively mitigate, detect, and protect systems against the highly evolving pace of zero-day malware attacks. It is against this backdrop that this paper presents a better approach to detecting and preventing zero-day malware attacks on mobile systems with Android in focus. The rest of this paper is articulately organized as follows; The introduction, related literature, research design, the result evaluation, discussion of result, conclusion and finally, recommendation for further studies.

RELATED LITERATURE

There have been significant research efforts on the problem of mobile malware detection. Generally, malware detection systems employ different approaches: Static analysis approaches such as (Aubery-Derrick, 2011; Christodorescu & Jha, 2003; Raymond *et al.*, 1995) are based on comparing applications to already known malware through a reverse engineering method that decompiles packaged applications and looking for signatures or using other heuristics within the program code. Other approaches like (Bryan *et al*., 2011; Hahnsang*et al*., 2008; Lei *et al*., 2009) monitor the power usage of applications, and report anomalous consumption. (Tchakounté, & Dayang, 2013; Burguera et al., 2011; Liang *et al*., 2010) used a dynamic analysis by monitoring system calls and attempt to detect unusual system call patterns. Some others like (Yajin*et al*., 2012; Abhijit*et al*., 2018) used the universal signature-based approaches that compare applications with known malware or other heuristics.

Burguera *et al*., (2011) presented Crowdroid a machine learning-based framework that recognizes Trojan-like malware on Android Smartphones, by analyzing the number of times each system call has been issued by an application during the execution of an action that requires user interaction. A genuine application differs from its trojanized version, since it issues different types and a different number of system calls. Crowdroid builds a vector of m features (where m is the number of the Android system calls). Crowdroid used about 100 system calls with only two trojanized applications tested. Mutz*et al*. (2006), presented a similar approach which also considered the system call parameters to discern between normal system calls and malicious ones. Asaf*et al.*, (2011) presented Andromaly that relies on machine learning techniques which monitors both the Smartphone and user's behaviours by observing several parameters, spanning from sensor activities to CPU usage. Andromaly used 88 features to describe applications behaviours; the features are then pre-processed by feature selection algorithms. The authors developed four malicious applications to evaluate the ability to detect anomalies. Other approaches only monitor anomalies on a limited set of functionalities such as incoming/outgoing traffic (Damopoulos *et al.*, 2011), SMS, Bluetooth and instant messaging (Abhijit& Shin, 2006), or power consumption (Jacobym *et al*., 2006), and therefore, their detection accuracy is high.

Xie *et al*., (2010) presented pBMDS; Propose Behaviour-based Malware Detection System (pBMDS) that correlates user's inputs with system calls to detect anomalous activities related to SMS/MMS sending. Abhijit *et al.*, (2018) propose behavioural detection framework to detect mobile malware, instead of common signature-based solution currently available for use in mobile devices. They represent malware behaviours based on a key observation that the logical ordering of application actions over time often reveals the malicious intent even when each action alone may appear harmless. Also, they propose a two-stage mapping technique that constructs malicious behaviour signatures at run-time from the monitored system events and API calls while studying 25 distinct families of mobile malware in Symbian OS. They discriminate the malicious behaviour of malware from the normal behaviour of applications by training a classifier based on Support Vector Machines (SVM). Detection rates from simulated and real malware samples were stated to be 96%.

Finally, Shabtai *et al*., (2010) presented a methodology to detect suspicious temporal patterns as malicious behaviour, known as knowledge-based temporal abstraction. Although their approach is recommended for detecting continuous attacks (e.g. DoS and worm infection), it lacks the ability to detect Trojan Horses the most frequently seen attacks nowadays. According to Christodorescu, (2007), Anomaly or Behaviour-based approaches are better in detecting 'zero-day' attacks compared to signature based approaches hence, he opined that Signature-based scanning must be supplemented with powerful host-based agent that employs behavioural analysis. While intrusion detection models with host-based data collection provide more accurate and reliable results than other approaches (Markus *et al*., 2006). And a host-based architecture have access to private information on the mobile device that is useful to detect intrusions as the information collected from the mobile device will reflect the device behaviour accurately.

Table 1 shows the list of monitored features used in this work. To generate a good feature vector that represents typical Android applications behaviour the design of the system utilized features that represents behaviours when the device is active and when it is inactive.

Table 1: List of Monitored Features

S/No.	Features
1	In/Out SMSs (two features).
2	In/Out Calls (two features).
3	Device Status (one feature).
4	Running Applications/Processes (one feature).
5 .	Date/Time Stamp (one feature).

RESEARCH DESIGN

The Anomaly Android Malware Detection System is composed of six modules which work together to provide the resources and mechanisms needed to detect malware on the Android platform. Each module has a specific functionality within the system. The integration of all the modules forms the System. Figure 1 shows an elaborate research design.

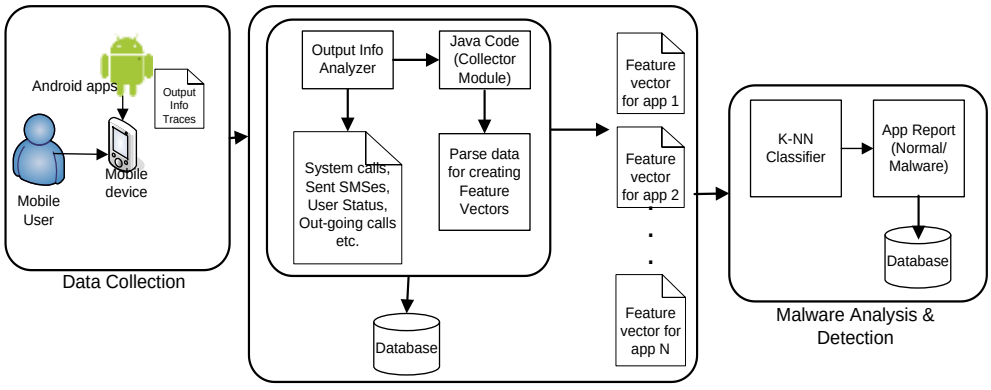

Fig. 1: System Design

In order to collect the data required for analysis, three monitor modules are implemented at the applications layer namely the *call monitor* module which will record all outgoing and incoming calls initiated by the device over a period of time T and forward the collected data to the *collector* module. The *SMS monitor* module also records all sent and received SMSs over a period of time T and forward the monitored data to the *collector* module. The *Statusmonitor* module monitors and records the device status. The device operates in two modes; Idle (hibernated or device screen in OFF) mode or active (device screen in ON) mode. The *collector* modules which is described in detail in subsequent subsection is responsible for collecting data from all the monitor modules and parse them into sets of feature vectors. In essence, the *collector* module will be

responsible for creating feature vectors from the monitored data and storing these vectors in the *logger* module. The *logger* module will receive the sets of feature vectors from the *collector* module and store them as local files in folder on the SD memory card for the *classifier* to finally classify as either normal or malicious application using the K-NN algorithm. The results of classification or labelling by the classifier module are also stored in the logger module which can be easily accessed or retrieved.

The Malware Detection Process

The malware detection processes are divided into three major activities namely;

i. Data Collection: This activity will allow for the collection of applications data from device through the implementation of the different monitors described in Figure 1.

ii. Data Processing: This activity comprises of managing and parsing all of the data collected from Android device into feature vectors. The data analyser code implemented in the *collector* module of the detection system collects the extracted data by the monitors for the monitored application and analyse them by converting them into the desired format which is the .arff.

iii. Malware Analysis and Detection: This is the final activity that is carried out by the detection system and it consists of analysing and classifying the feature vectors of applications obtained in data processing phase in order to create the test feature vectors which are then analysed for anomaly behaviours to detect malicious or anomalous behaviour in the Android applications. The feature vectors are classified into two different classes of "Normal" or "Malicious" using the K-NN classifier in Weka. This algorithm will create two classes after it is trained using the normality model. All feature vectors belonging to good applications are classified into the "Normal" class while feature vectors belonging to malware applications are classified into and the malicious class.

The Normality Model

In order to efficiently develop a machine learning model, it is important to train the model on the normal and abnormal behaviour of the system. To do this, a normality model is required to describe to the classifier the pattern of behaviours. Hence a normality model is designed based on the fact that malware requires user interaction to activate its payload on the target device. For malware that uses SMS and calls as its propagation vector, it becomes evident that user interaction is necessary for such malware to propagate. SMSs and calls require user interaction with the device to compose and send SMSs or to initiate calls. Therefore, a normal SMS and call activity is one that has active user interaction. In this work, five (5) features were used to describe a normality model for the K-NN model, these features include:

i. The out-going call
ii. The In-coming call
iii. The Out-going SMS
iv. The In-coming SMS and
v. The device Status.

These features were used as follows;

i. If the device is active or inactive at the point of any activity;
ii. If any SMS is being sent or received when the phone is inactive and
iii. If any call is being made or received when the phone is inactive.

The classification task involved in this work is a binary classification in the sense that there are two classes; Normal and Malicious class. Using a binary representation for the features, the number of probable permutations of these 5 features is obtained by the expression given as

$$2^n$$

Where n is the number of features to be represented.

Since in this case n = 5; the expression yields $2^5 = 32$ instances of the features as given in Table 2. The value of 1 represents the presence of the feature while the value of 0 represents the absence of that feature. The numeric count of how many occurrence of the feature is immaterial because even a single presence is enough to describe the entire behaviour. For the device status, 1 represents an active user interaction where the device screen is 'ON' and 0 represents no interaction with the device with the screen turned 'OFF' or hibernated. The combination of the behaviour features gives thirteen (13) normal instances and nineteen (19) malicious instances based on the condition that certain activity do not occur at the same time and at certain device state. For example Table 2 gives the instances and possible classification result.

Table 2: Instance Classification

S/N	OutCall	InCall	OutSMS	InSMS	Device Status	Class
1	0	0	1	0	0	Malicious
2	0	0	1	0	1	Normal

The first instance signifies the occurrence of an out-going SMS while the device screen is in an inactive (OFF/hibernated) state. The application behaviour represented by this instance is suspicious; the reason is that sending SMS requires active interaction with the device; to compose the text message and then send it by pressing the send button. This activity would not have been made by a valid user when the device is idle hence; it is classified as a malicious behaviour. In the second instance, the out-going SMS occurred while the device screen state is active signifying that there is active interaction with the device which keeps the screen light 'ON' hence, the activity is classified as a normal activity. This normality model is parsed and converted into arff with the date/time stamp and application and or services features appended to each instance and then used to train the K-NN classifier.

System Design

This subsection presents the designs for the various components of the system which includes; interface design, input and output design and other related subsystems.

Input/Output Designs

The input to the classifier is the result of monitoring extracted from the applications by the various monitors. This forms the test set for the classifier function. The data extracted by the monitors from the applications during execution are logged in a file in csv format. This file is the first output from the monitoring modules which in turn serves as the input to the collector module. When this file gets to the collector module, it is processed into arff hence; the output of the collector module is an unclassified arff file. This unclassified arff file is the input to the logger module as well as the classifier module. For the logger module, its inputs are two; first the unclassified arff file from the collector module and secondly, the classified arff file which is the result of classification from the classifier module is sent to the logger for storage. The classified arff file is therefore the output of the classifier module.

Interfaces Design

The interfaces are the Graphical user Interface (GUI) which presents visual display of the system to the users. Figure 2 presents the interface for the malware detection system. Other interfaces representing the subsystems are not given here for lack of space.

Fig. 2: Screen Capture of the Malware Detection System Interface.

MATERIALS USED

The detection framework was implemented on a laptop machine with the Intel Core-i3-370M Processor, 3GB of available memory and 500GB Hard Disk Drive (HDD). This machine runs Windows 7 Operating System and tests werecarried out on a TECNO P5 with build number P5-G255-20140313, Android Jelly Bean version 4.2.2 OS, and Linux kernel version 3.4.5. The implementation does not require rooting or jail breaking of the device since the monitored features are all carried out at the application layer. The component of the system framework includes an Android Application in Java implemented using the Android Studio version 1.3.2 Integrated Development Environment (IDE) as the Software Development Kit (SDK). This tool runs Dalvik which is a virtual machine for Android. Android Studio is a java based development tool that provides a professional-grade development environment for building Android applications. It is a full Java IDE with advanced features to help build, test, debug, and package Android applications with a background Dalvik virtual machine. In order to realise the classification model of the K-NN classifier, Weka version 3.7.3; an open source library in Java that includes several classification tools was used by adding the Weka.jar file as an external library to the Android Studio project from where the available features were invoked programmatically using sets of available Java APIs.

RESULT EVALUATION

The results obtained from the test are shown in Figure 3, Figure 4 and clearly tabulated and summarized in Table 3. These results were obtained from a single run of the detection model and are discussed based on the evaluation measures discussed here.

```
=== Run information ===
Scheme:       weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A
\"weka.core.EuclideanDistance -R first-last\""

Relation:     AttributeFeatures-weka.filters.unsupervised.attribute.StringToNominal-R2
Instances:    32
Attributes:   8
              Time
              AppName
              Incall
              OutCall
              InSMS
              OutSMS
              Screen
              Class

Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===
IB1 instance-based classifier
using 1 nearest neighbour(s) for classification

Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          30               93.75   %
Incorrectly Classified Instances         2                6.25   %
Kappa statistic                          0.8672
Mean absolute error                      0.215
Root mean squared error                  0.2618
Relative absolute error                 44.2467 %
Root relative squared error             53.0226 %
Coverage of cases (0.95 level)         100      %
Mean rel. region size (0.95 level)      87.5    %
Total Number of Instances               32

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.154    0.905      1.000    0.950      0.875    0.996     0.995     Normal
                 0.846    0.000    1.000      0.846    0.917      0.875    0.996     0.989     Malicious
Weighted Avg.    0.938    0.091    0.943      0.938    0.936      0.875    0.996     0.993
```

Figure 3: Detailed Results of the Test Performed

```
=== Confusion Matrix ===

   a  b   <-- classified as
 | 19  0 |  a = Normal
 |  2 11 |  b = Malicious
```

Figure 4: The Confusion Matrix

The confusion matrix is of the form:

$$\begin{bmatrix} w & x \\ y & z \end{bmatrix}$$

This is a 2 x 2 matrix representing the two classes (a = Normal and b = Malicious) where the entries w = nNN; x = nNM; y = nMN and z = nMM.

The Confusion matrix of Figure 4 shows the misclassified malicious samples and the correctly classified samples from the experiment, the incorrectly classified cases were due to the malicious class samples misclassified as being of the Normal class. It should be noted from Figure 3 that the time taken to build the model is less than a second, this execution time parameter shows that the K-NN classification model and the current dataset yields very promising results for its applicability on real-time monitoring of malware infections on real Android devices.

Table 3: Summary of Results

Class	Accuracy	Error Rate	TPR	TNR	FPR	FNR	ρ	AUC	Recall	Sensitivity	Specificity
Normal	0.9375	0.0625	1.000	1.000	0.154	0.154	0.905	0.996	1.000	0.9048	1.000
Malicious	0.9375	0.0625	0.846	0.846	0.000	0.000	1.000	0.996	0.846	0.9048	1.000

DISCUSSION OF RESULTS

Based on the output of the test carried out, it is obvious that the K-NN classification model provides a very high accuracy of 0.9375 representing 93.75 percent (≈ 94 percent) of the samples correctly classified with error rate of as low as 0.0625 representing 6.25 (≈ 6) percent as shown in the output of Figure 4 and summarized in Table 3. The TPR of the Normal and Malicious samples which are the same as the Recall are 1.000 and 0.846 while the precisions are 0.905 and 1.000 respectively. The precision of the samples classified as Normal and the samples classified as Malicious did not vary much from each other meaning that the predictive capacity of the K-NN classifier is almost equal in both cases.

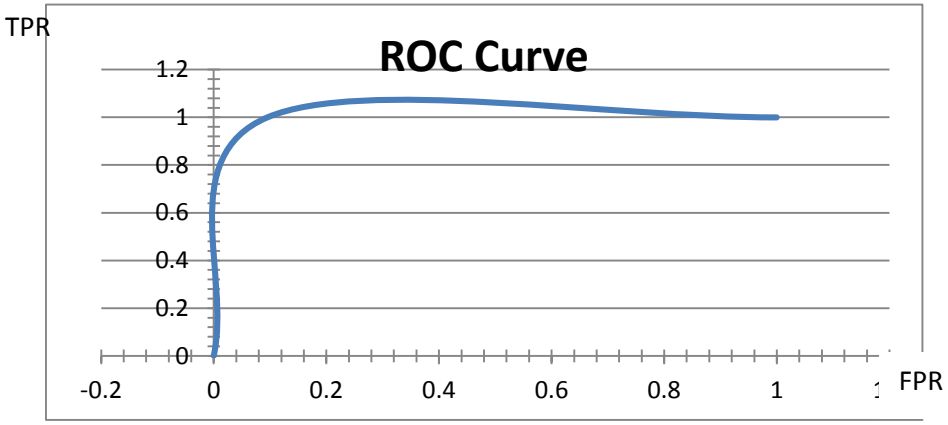

Fig. 5: The Receiver Operating Characteristic (ROC) Curve

The Area Under Curve (AUC) of the ROC is 0.996. The AUC has a standard range of $0 \leq AUC \leq 1$, which mean that the obtained value of 0.996 is a good indication of the performance of the K-NN classifier as a model for malware detection. As earlier stated, a perfect classifier will have an AUC of 1. Thus, the closer the AUC is to 1, the greater the classifier's predictive strength and hence the performance. Figure 6 shows the ROC curve which is a plot of FPR on the X-axis against TPR on the Y-axis. The ROC curve could also be represented as a plot of (1 – Sensitivity) against Specificity. The sensitivity and specificity measure of K-NN algorithm based on equations (4.9) and (4.10) are 0.9048 (90.48 percent) and 1.000 (100 percent) respectively and are the same for both classes. The sensitivity and specificity of the K-NN algorithm is very high as indicated. Sensitivity is the proportion of actual positive cases which are correctly identified while Specificity is the proportion of actual negative cases which are correctly identified.

These results were in conformity with results obtained by previous researchers in their works for instance, Su, *et al*, (2012) in their work using J48 decision trees and Random forest classifiers produced accuracies of 91.6% and 96.7% respectively. Similarly, Ali, *et al.*, (2013) in their study of machine learning classifiers for anomaly-based mobile botnet detection using K-NN produced 99.9% accuracy. While the researchers take cognizance of the differences in platforms, datasets, approaches and dimensions of their works to theirs, the performance results obtained in all cases bear close resemblance to each other without much difference and in some cases, the K-NN model performs better than other classifiers like the J48

decision trees attesting to the inherent high performance of the K-NN classification model. As earlier noted the performance of the classifier depends largely on the training set; the better the training set supplied to a classifier, the better the performance of that classifier.

CONCLUSION

This research presents a behaviour-based approach rather than a signature-based technique, this makes it possible to detect new and unknown malware based on their behaviours rather than their signature string which are not yet discovered. Hence, in this work, we have developed an intrusion detection system which was able to detect new and unknown attacks using Behavioural approach. This strengthens and re-affirm the assertion by Christodorescu, (2007) that Behaviour-based approaches are better in detecting zero-day attacks compared to signature based approaches. He further opined that signature-based scanning must be supplemented with a powerful host-based agent that employs behavioural analysis. Confirming the capability of anomaly or behaviour-based detection approach to detect unknown malware which signature-based system cannot detect. Our experiment also demonstrated the possibility of capturing Android system activities which served as behavioural features used for behavioural analysis. This implies that more fine-grain features can be derived even at a much lower level of system granularity to define system behaviours for behavioural analysis. A well-reviewed literature that adds to the available body of knowledge was also presented by this paper.

This paper has also by every means provided a novel approach to malware detection specifically, zero day attacks. The results obtained shows a very high accuracy of 0.9375 representing 93.75% of the samples correctly classified with error rate of as low as 0.0625 representing 6.25%. The TPR of the Normal and Malicious samples which are the same as the Recall are 1.000 and 0.846 representing 100% and 84.6% respectively while the precisions are 0.905 and 1.000 representing 90.5% and 100% respectively. The precision of the samples classified as Normal and the samples classified as Malicious did not vary much from each other meaning that the predictive capacity of the K-NN classifier is almost equal in both cases. The Area under Curve (AUC) of the ROC is 0.996 representing 99.6%.These shows that behavioural approach with KNN is capable of malware detection with a very high degree of accuracy.

RECOMMENDATIONS FOR FURTHER STUDIES

System behaviours such as Mobile device activities can be extracted at low level of granularity for the purpose of behavioural detection. Currently, access to kernel layer data is deprecated on modern versions of Android hence, accessing kernel layer data is not possible without routing the device. Hence, the following are recommended for further studies; the dataset which is the set of monitored features need to be expanded beyond SMSs, calls and device status to incorporate more features that could give a more general and system wide representation of the behaviour of the system; access to low level information like system calls, network traffic and other system level information which are presently deprecated in Android system should be allowed access by Google in some way that would not require rooting, and this framework should be extended beyond Android to include other mobile devices Operating Systems (iOS, RIM, Symbian, Java and Windows).

REFERENCES

Abhijit, B. & Shin, K.G. (2006). Proactive Security for Mobile Messaging Networks. In ACM Workshop on Wireless Security, WiSe '06, pp. 95-104.

Abhijit, B., Xin H., Kang, G.S., & Taejoon, P. (2008). Behavioural Detection of Malware on Mobile Handsets. In Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services, MobiSys ’08, pp. 225–238.

Ali, F., Nor, B.A., Rosli, S., Fairuz, A., Rauf, R.M., & Shahaboddin, S. (2013). A Study of Machine Learning Classifiers for Anomaly-based Mobile Botnet Detection. Malaysian Journal of Computer Science, 26(4), pp. 251-265.

Asaf S., Uri K., Yuval E., Chanan G., & Yael W. (2011). Andromaly: A Behavioural Malware Detection Framework for Android Devices. Journal of Intelligent Information Systems, pp 1-30. DOI: 10.1007/s10844-010-0148-x.

Aswathy, D. (2013). An Analysis of Mobile Malware and Detection Techniques. pp 1- 13. Retrieved from http://www.cs.tufts.edu/comp/116/.../adinesh.pdf visited 10th March, 2014.

Aubery-Derrick S. (2011). “Detection of Smart Phone Malware”, Electronic and Information Technology University Berlin Unpublished PhD. Thesis. PP. 1-211.

Bryan, D, Yifei, J., Abhishek, J. & Shivakant, M. (2011). Location Based Power Analysis to Detect Malicious Code in Smartphones. In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, SPSM ’11, pp. 27–32.

Burguera, I., Zurutuza, U. & Nadjm-Tehrani, S. (2011). Crowdroid: Behavior-based Malware Detection System for Android. In Proceedings of the 1st ACM workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 15-26.

Christodorescu, M. (2007). Behaviour-based Malware Detection. Unpublished Ph.D Thesis, Computer Science and Engineering, University of Wisconsin-Madison, August 2007, 1-54.

Christodorescu, M. & Jha, S. (2003). Testing Malware Detectors. In Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA’04), July

11-14, 2004, Boston, Massachusetts, USA. 34-44. doi: 10.1145/1007512.1007518.

Damopoulos, D., Menesidou, S.A., Kambourakis, G., Papadaki, M., Clarke, N. and Gritzalis, S. (2011). Evaluation of Anomaly-based IDS for Mobile Devices Using Machine Learning Classifiers. John Wiley & Sons, Ltd. Security and Communication Networks 2011; 00:1-9. doi:10.1002/sec

Denis, M. (February, 2012). Mobile Malware Evolution, Part 5. Securedlist, pp. 1. Retrieved from http://www.securelist.com/en/analysis/204792222/Mobile_Malware_Evolution_Part_5

Gartner, (November 2015). Worldwide Smartphone Sales to End Users by Operating System in 3Q15. Gartner Report Retrieved from http://www.smartphonemarketresearch.com/emerging-markets-drove-worldwide-smartphone-sales-to-15-5-percent-growth-in-third-quarter-of-2015/ visited 20th January, 2016.

Hahnsang, K., Joshua, S. & Kang, G.S., (2008). Detecting Energy Greedy Anomalies and Mobile Malware Variants. In Proceedings of the 6th international Conference on Mobile Systems, Applications, and Services, MobiSys '08, pp. 239–252.

Joshua, A., Waziri, O.V., Abdullahi, M.B., Ume, U.A. & Adewale, O.S., (2015). Extracting Android Applications Data for Anomaly-based Malware Detection. Global Journal of Computer Science and Technology (E) Network, Web and Security (GJCST-E), 15(5): Version I, pp. 1-8.

Jacobym G.A., Marchany R., Davis N.J. IV (2006). How Mobile Host Batteries Can Improve Network Security. IEEE Security and Privacy Vol. 4 PP. 40-49.

Lei, L., Guanhua, Y., Xinwen, Z. & Songqing C. (2009). Virusmeter: Preventing your Cell Phone from Spies. In Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection, RAID '09, pp. 244–264.

Liang Xie, Xinwen Zhang, Jean-Pierre Seifert, and Sencun Zhu, (2010). PBMDS: A Behavior-based Malware Detection System for Cell Phone Devices. In Proceedings of the third ACM conference on Wireless network security, WiSec '10, pp. 37–48.

Lovi D. & Divya, B. (2014). Taxonomy: Mobile Malware Threats and Detection Techniques. Dhinaharan Nagamalai (Eds) : ACITY, WiMoN, CSIA, AIAA, DPPR, NECO, InWeS2014 pp. 213–221.

Markus, M., Perttu, H. & Kimmo, H. (2006). Host-Based Intrusion Detection for Advanced Mobile Devices, In IEEE 20th International Conference on Advanced Information Networking and Applications, 2006. AINA 2006. 2, 72-76, doi:10.1109/AINA.2006.192

Mutz, D., Valeur, F., Vigna, G. (2006). Anomalous System Call Detection. ACM Transactions on Information and System Security 9(1), 61-93.

Nwokedi, I. & Aditya, P.M. (2007). A Survey of Malware Detection Techniques. Unpublished Predoctoral Fellowship and Purdue Doctoral Fellowship Research Report, Department of Computer Science, Purdue University, West Lafayette IN 47907. pp. 1-48.

Oberheide, J., Evan, C. & Farnam, J. (2008). CloudAV: N-version Antivirus in the Network Cloud. In Proceedings of the 17th USENIX Security Symposium (Security '08), San Jose, CA, July 2008.

Raymond, W.L, Karl, N.L. & Ronald, A.O. (1995). MCF: A Malicious Code Filter. Computers and Security, 14(6), pp. 541 – 566.

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C. & Weiss, Y. (2010). Andromaly: A Behavioural Malware Detection Framework for Android Devices. Journal of Intelligent Information Systems, pp. 1-30, doi: 10.1007/s10844-010-0148-x.

Spamlaws, (2017). Zero Day Attacks and How to Prevent Them. http://www.spamlaws.com/zero-day-attacks.html visited 21st June, 2017.

Srikanth, R. (2012). Mobile Malware Evolution, Detection and Defense, EECE 571B Unpublished Term Survey Paper, Institute for Computing, Information and Cognitive Systems, University of British Columbia, Vancouver, Canada, April, 2012, pp. 1-4. Retrieved from http://www.cs.tufts.edu/../adinesh.pdf visited 2nd April, 2014.

Statista, (2015). Global Smartphone Sales 2009-2014, by OS. Retrieved from http://www.statista.com visited 11th November, 2015.

Su, S., Chuah, M. & Tan G., (2012). Smartphone Dual Defense Protection Framework: Detecting Malicious Applications in Android Markets, Proceedings of the 2012 8th International Conference on Mobile Ad hoc and Sensor Networks, Chengdu, China, pp. 153-160.

Tchakounté, F. & Dayang, P. (2013). System Calls Analysis of Malwares on Android. International Journal of Science and Technology 2(9), pp. 669-674.

Xie, L., Zhang, X., Seifert, J.P. & Zhu, S. (2010). pBMDS: A Behavior-based Malware Detection System for Cell Phone Devices. In: Proceedings of the Third ACM Conference on Wireless Network Security, WISEC 2010, Hoboken, New Jersey, USA, March 22-24, 2010, pp. 37-48.

Yajin, Z., Zhi, W., Wu, Z. & Xuxian, J. (2012). Hey, you, get off of my Market: Detecting malicious Apps in Official and Alternative Android Markets. In Proceedings of the 19th

Network and Distributed System Security Symposium, 2012, pp. 44.

You, J.H., Daeyeol, M., Hyung-Woo, L., Jae, D.L. & Jeong, N.K. (2014). Android Mobile Application System Call Event Pattern Analysis for Determination of Malicious Attack. International Journal of Security and Its Applications 8(1), pp. 231-246. http://dx.doi.org/10.14257/ijsia.2014.8.1.22 Visited 9th February, 2015.