# A network-based framework for mobile threat detection

Sanjay Kumar*, Ari Viinikainen† and Timo Hamalainen‡
Faculty of Information Technology
University of Jyvaskyla
Jyvaskyla, Finland
Email: *sanjay.k.kumar@jyu.fi, †ari.viinikainen@jyu.fi, ‡timo.t.hamalainen@jyu.fi

***Abstract*—Mobile malware attacks increased three folds in the past few years and continued to expand with the growing number of mobile users. Adversary uses a variety of evasion techniques to avoid detection by traditional systems, which increase the diversity of malicious applications. Thus, there is a need for an intelligent system that copes with this issue. This paper proposes a machine learning (ML) based framework to counter rapid evolution of mobile threats. This model is based on flow-based features, that will work on the network side. This model is designed with adversarial input in mind. The model uses 40 time-based network flow features, extracted from the real-time traffic of malicious and benign applications. The proposed model not only to detects the known and unknown mobile threats but also deals with the changing behavior of the attackers by triggering the retraining phase. The proposed framework can be used by the mobile operators to protect their subscribers. We used several supervised ML algorithms to build the model and got an average accuracy of up to 99.8%.**

***Index Terms*—Intrusion Detection, Mobile Threats, Machine Learning, Concept-drift, Anomaly detection**

## I. Introduction

According to Statista [1], a number of mobile users are rising rapidly, and it will surpass 5 billion by 2019. Nowadays, people use smart-phones for a variety of purposes from online shopping to bank transaction, which makes the smart-phones attractive target to cybercriminals. According to the IDC [2], 85% of the smart-phones running Android and thus it becomes the most target-able OS.

According to MCafee threat report [3], mobile threats are evolving with the nature of the mobile devices. These threats are not limited to smart-phones, but also target the Internet of things (IoT) devices which are controlled by smart-phones apps. Some of the common mobile threats include spy-ware, ransom-ware, banking Trojans, premium messages senders, and private information stealing.

The protection systems that use signatures to detect threats are good at identifying known attacks [4], but the minor modification can easily bypass these signature-based systems [5]. Nowadays, cybercriminals use a variety of methods to evade detection from traditional intrusion detection systems (IDS) and anti-virus systems. In our analysis, we have also observed that many malicious applications were using Google Docs for the data transmission and these packets can be easily bypassed through the traditional systems. Encrypted traffic also hurdles the deep packet inspection and needs more computational resources [6]. Shallow Packet inspection can combat several issues caused by encrypted traffic [7].

In the mobile threats, the weakest link is the user. According to Nokia [8], 71% mobile users do not install anti-virus system and the anti-virus systems are not capable of detecting these sophisticated threats. Another issue with the traditional anti-virus system is that the adversary can try their evasion techniques.

Therefore, there is a need for a network-based solution, which can protect users from these advanced threats efficiently, while retaining the privacy of the user. The machine learning and artificial intelligence offer powerful tools to cope with this issue. In this paper, we propose a network-based framework to detect mobile threat using machine learning models. Our proposed model uses 40 time-related network traffic features (see Table 1) to identify known/unknown attacks. This detection model uses TCP flows of the network traffic, which includes several bidirectional time-based features. These time-based features are bi-directional flows of malware communication. The features used in this study are different from many flow-based techniques proposed for network-based intrusion detection systems, due to their characteristics. We have used several ML algorithms to train and test our model, that include J.48, random forest (RF), multilayer perceptron (MLP), radial basis function (RBF) and Deeplearning4j (DL4J). The ML-based anomaly detections systems often need regular re-training in a non-stationary environment and adversary can poison the system. The proposed model also requires re-training, but it is not possible for the adversary to poison the system by forcing the system to learn the patterns according to them.

The main contribution and novelty of this paper include the proposed network-based ML classification model that uses 40 time-related network flow based features to detect known/unknown threats. These bi-directional features are unique, as in best of our knowledge these features were not used by any other study at the time of this research. This model also comprises a retraining phase to avoid concept-drift situation. This model can detect known and unknown threats and classify them according to the family they belong. The features used in this research were extracted from real-time traffic generated by several malicious and benign applications. We have developed this dataset due to the non-availability of any latest labeled public dataset that applies to mobile threats. The time-based

bidirectional flows of malware communication are used in this research, that makes it different from several other studies. The proposed solution is also unique in the way that it also comprises the retraining phase to reduce the chance of concept drift. This framework can deal with the changing behavior of the attacker by initializing the re-training phase which can increase the performance of the classifiers.

The structure of this paper is as follows. In Section 2, we discuss the previous work in Network-based mobile intrusion detection systems and their limitations. Section 3 describes the methodology, proposed ML model/framework and the development steps. Section 4 is based on the experiments conducted in this study and the obtained results from the machine learning classifiers. Finally, in Section 5 the conclusion and future work of this research are outlined.

## II. Related Work and their limitations

Artificial intelligence and machine learning are becoming prevalent in the field of cybersecurity. Most studies in the area of Android malware detection, have only focused on features such as system or API calls. Some of the solutions proposed for mobile threat detection such as [9]–[12], are device-based and need to be installed on the mobile device, like a traditional anti-virus. However, many users do not install security applications on their devices. Due to ignorance of the users, there is a need for an efficient network-based mobile threat detection system. Several researchers such as [13], [14] focused on network-based solutions to detecting mobile malware and intrusions and produced effective results. Drebin [10], a very popular method of detecting android malicious application, uses SVM to classify applications using several features such as Permissions, API calls, or Network address.

In 2014 Narudin et al. [9] proposed a model of detecting Android-based malware using ML classifiers. The research was carried out on the samples from the MalGenome [15] dataset and produced good results on the know data, i.e., 99.7% true positive rate (TPR), while the significant decrease was seen on unknown data with TPR of 74.5%. In 2014, a mobile botnet detection model was proposed that uses TCP Size, connection duration, and GET/POST parameters detect botnet [11]. The features were extracted from 100 botnet samples. The study produced TPR of 99.94% and false positive rate (FPR) of 0.06% on the known data. The study used only 3 features and attackers change their traffic patterns, so there is high chance that it produces false positives or false negatives after an interval of time. In 2012, a framework to check malicious applications on the Android store was proposed by Su et al. [12] and it used J.48 and Random Forest for the classification of the malicious applications. The system checks the system calls and if any anomaly found in the system calls then the traffic is generated from the sample to do packet inspection to make the decision [12] . All of these systems need to be installed on the device itself. Our proposed system works on the network side, which is the black box to the attacker. Several researchers [16]–[18] used flow-based techniques to detect botnet. Flow-based techniques are extremely useful in detecting threats [7], as most of the malware communication is encrypted [6]. Flow-based techniques also proven to be successful in detecting website fingerprinting attacks [19], [20].

## III. Methodology

We proposed a network-based framework to detect mobile threat using Machine Learning techniques. By analyzing the time-related flow-based features of the network traffic, it can detect malicious network patterns. Fig. 1 shows the process of the model. The model was developed in several phases. The first phase was sample collection and traffic generation from the malicious and benign application. In the second phase, 40 time-based features were extracted from the bi-flows of the traffic generated by these applications and then labeled accordingly. After that, the model was built using several ML algorithms. This model is different from many other studies as it uses many different time-related features and it has a retraining phase to deal with concept drift situation. It is usually seen that adversary change the attack patterns and evasion techniques with the advent of time and the ML models produce false positives or false negatives. Our framework is capable of coping with this kind of situations. There is another advantage of this network-based framework, the algorithm and features used in the system are unknown to the adversary. So, it will be hard for the attacker to try their evasion techniques in comparison to traditional antivirus'. Also, the concept-drift function is designed in a way that the "Poisoning" by the adversary could not be possible.

### A. Sample collection and Traffic Generation

We have used the same sample collection and traffic generation method as used in our last paper [13]. Traffic was generated for both benign and malicious applications. Virustotal [21] was used to download a number of malware samples while the benign samples were collected through Google playstore.

*1) Normal Traffic Generation:* Several benign applications were downloaded in different android virtual machines. These applications were executed at a different interval of time. Traffic generated by these virtual machines was captured and features were extracted from the traffic flows. Table I shows the benign applications used in this study.

*2) Malicious Traffic Generation:* The traffic used in this study was generated by the method mentioned in our previous work [13]. We have used some previously generated traffic and also generated new traffic for the testing purposes. We have used around 700 malicious samples of several malware families to generate our malicious traffic dataset. Virustotal was used to download these samples using several conditions. A public sandbox "Anubis (Andrubis)" [22] and "Cuckoo" [23] was used to generate the traffic.

### B. Preprocessing

In the Preprocessing, the PCAPs of the generated traffic were converted into meaningful datasets, by extracting features

TABLE I: Benign Applications downloaded from Google App Store

| S.No | Google App | Action Performed |
|---|---|---|
| 1 | Facebook Messenger | Login with a valid account and sent messages |
| 2 | Facebook | Login with a valid account, visited several timelines, did comments etc |
| 3 | Gmail | Login with a valid accounts, sent and received emails. |
| 4 | Google Maps | Searched for some locations |
| 5 | Twitter | Login with a valid account and performed several actions |
| 6 | Youtube | Watched some videos |
| 7 | Chrome | Navigate through several webpages |
| 8 | Skype | Login with a valid account. Skype video chat and sent messages. |

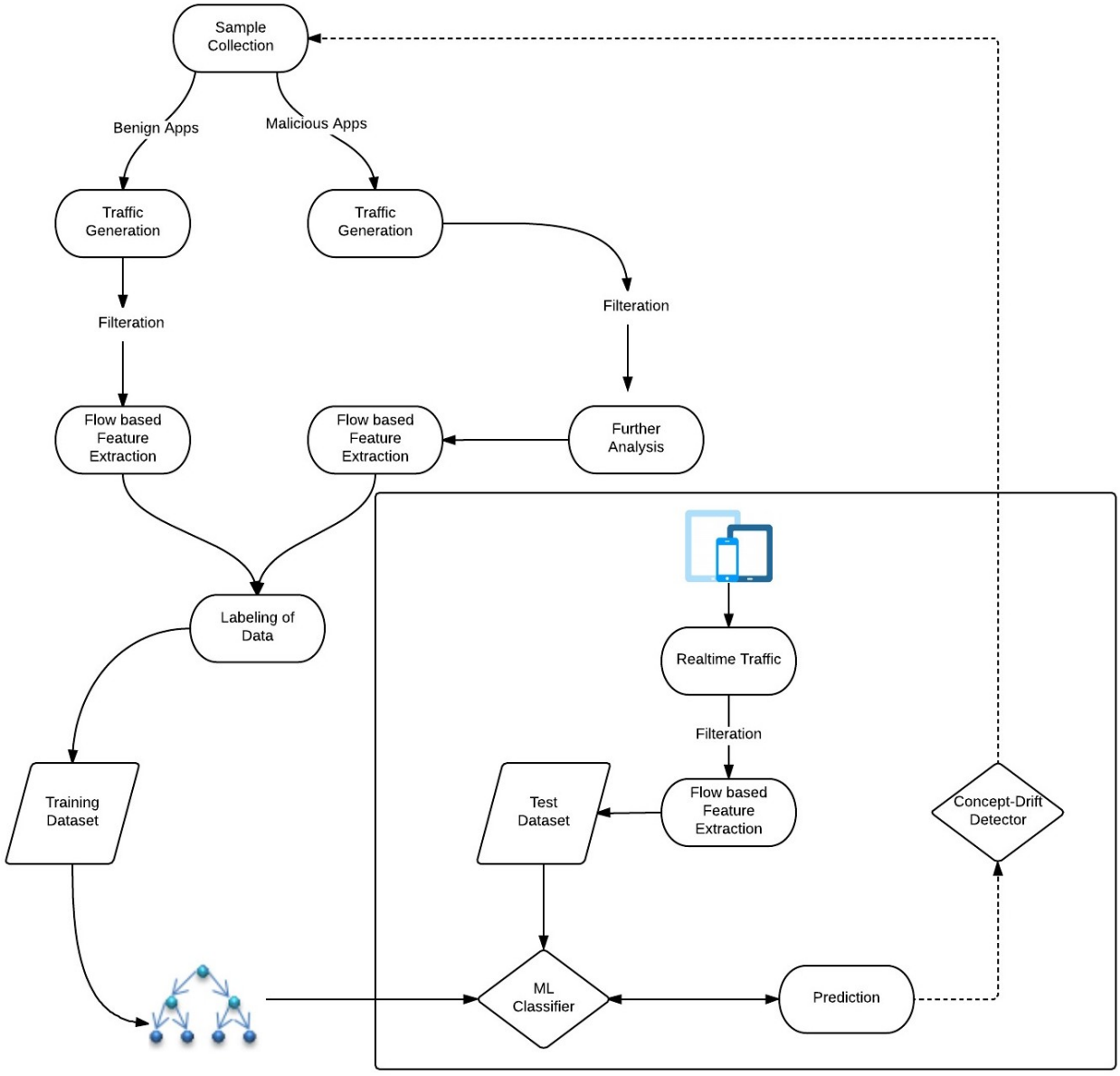


Figure 1: Machine Learning model for Mobile threat detection

TABLE II: Feature List

| Feature No. | Feature | Description |
|---|---|---|
| 1 | DstPort | The destination port number |
| 2 | DurationmSec | The duration of the flow (in microseconds) |
| 3 | StoDPackets | No. of packets in source to destination |
| 4 | StoDBytes | Bytes sent from source to destination |
| 5 | DtoSPackets | No. of packets in destination to source |
| 6 | DtoSBytes | Bytes sent back from destination to sources |
| 7 | SmallStoDPktBytes | Smallest packet sent from Source to Destination |
| 8 | MeanStoDPktBytes | The mean size of packets sent from Source to Destination |
| 9 | LargeStoDPktBytes | Largest packet sent from Source to Destination |
| 10 | StDevStoDPktBytes | Standard deviation from the mean of the packets sent from Source to Destination |
| 11 | SmallDtoSPktBytes | Smallest packet sent from Destination to Source |
| 12 | MeanDtoSPktBytes | The mean size of packets sent from Destination to Source |
| 13 | LargeDtoSPktBytes | Largest packet sent from Destination to Source |
| 14 | StDevDtoSPktBytes | Standard deviation from the mean of the packets sent from Destination to Source |
| 15 | MinStoDmSec | Minimum time (in microseconds) between two packets sent from Source to Destination |
| 16 | MeanStoDmSec | Mean time(in microseconds) between two packets sent from source to Destination |
| 17 | MaxStoDmSec | Maximum time(in microseconds) between two packets sent from source to Destination |
| 18 | StDevStoDmSec | Standard deviation from the mean time (in microseconds) between two packets sent from source to Destination |
| 19 | MinDtoSmSec | Minimum time (in microseconds) between two packets sent from Destination to Source |
| 20 | MeanDtoSmSec | Mean time (in microseconds) between two packets sent from Destination to Source |
| 21 | MaxDtoSmSec | Maximum time (in microseconds) between two packets sent from Destination to Source |
| 22 | StDevDtoSmSec | Standard deviation from the mean time(in microseconds) between two packets sent from Destination to Source |
| 23 | AvgsFlowStoDPackets | The average number of packets in a sub flow from Source to Destination |
| 24 | AvgsFlowStoDBytes | The average number of bytes in a sub flow from Source to Destination |
| 25 | AvgsFlowDtoSPackets | The average number of packets in a sub flow from Destination to Source |
| 26 | AvgsFlowDtoSBytes | The average number of packets in a sub flow from Destination to Source |
| 27 | MinActivemSec | Minimum time (in microseconds) flow was active before going idle |
| 28 | MeanActivemSec | Mean time (in microseconds) flow was active before going idle |
| 29 | MaxActivemSec | Maximum time (in microseconds) flow was active before going idle |
| 30 | StDevActivemSec | The standard deviation from the mean time (in microseconds) that the flow was active before going idle. |
| 31 | MinIdlemSec | The minimum time a flow was idle before becoming active (in microseconds) |
| 32 | IeanIdlemSec | The mean time a flow was idle before becoming active (in microseconds) |
| 33 | MaxIdlemSec | The maximum time a flow was idle before becoming active (in microseconds) |
| 34 | StDevidlemSec | The standard deviation from the mean time a flow was idle before becoming active (in microseconds) |
| 35 | StoDPSHCount | No. of times PSH flag was set in packets traveling from Source to Destination |
| 36 | DtoSPSHCount | The number of times the PSH flag was set in packets traveling from Destination to Source |
| 37 | StoDURGCount | The number of times the URG flag was set in packets traveling from Source to Destination |
| 38 | DtoSURGCount | The number of times the URG flag was set in packets traveling from Destination to Source |
| 39 | TotalStoDHdrLen | The total bytes used for headers from Source to Destination |
| 40 | TotalDtoSHdrLen | The total bytes used for headers from Destination to Source |

from network flows and labeling them. Before the extraction of features, raw traffic was further analyzed to identify any normal traffic generated by malicious samples, as incorrect labels could lead to inefficient model. Several tools and techniques including an online PCAP Analyzer "*NetworkTotal*", were used to verify the PCAPs.

*1) Feature Extraction:* The features were extracted according to "RFC-5103 BiFlow Export" using IPFIX [24] and RFC 2724. According to [25], there are several advantages of using bi-directional flows in security analysis.

40 time-based features (see Table II) were extracted from each traffic flow, known as "Instance". "Flowtbag" was used to extract these features. The flow is a 5-tuple in which all the packets having the same Source IP, Destination IP, port numbers(source and destination ports) and protocols. These features have unique characteristics so that the attackers cannot change most of these traffic features to evade detection. It was also seen in our previous study [26], that using an ensemble of multiple ML classifiers can increase the performance and reduce the chance of concept drift.

*2) Labeling:* The instances were labeled as "Normal" or "Malicious" accordingly. We have also created a dataset in which the instances were labeled according to the malware family they belong.

### C. Machine learning classification model

The ML algorithms that were used in this research are RBF, Random Forest, DeepLearning4j, J.48, and MLP. The training of the ML model is based on the real-time traffic from the malicious and benign samples. We have used ten-fold cross-validation and percentage split for the training and testing the classifier. The classifiers were also tested on the new test dataset.

### D. Concept-Drift Detector

Adversary always tries to change their traffic patterns to evade detection, and therefore concept-drift occurs in machine learning methods. To avoid concept-drift, a retraining phase is introduced in our framework. As this solution is network based, so it is hard for the adversary to test their evasion techniques. Although, there are many studies that show limitation related to the security of ML models [27]–[30]. This concept-drift feature was designed while keeping these limitations in mind. The retraining phase will trigger according to several

TABLE III: Experiment 1: Cross validation

| Performance Evaluation on Dataset 1 using 10 Fold Cross Validation | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ML Algorithm | TPR | FPR | TNR | FNR | Accuracy | F1 Measure | Precision | Area Under ROC (AUC) |
| RBF | 1.000 | 0.105 | 0.895 | 0.000 | 0.996 | 0.998 | 0.996 | 0.983 |
| Random Forest | 0.999 | 0.004 | 0.996 | 0.001 | 0.999 | 0.999 | 1.000 | 1.000 |
| MLP | 0.999 | 0.021 | 0.979 | 0.001 | 0.998 | 0.999 | 0.999 | 0.999 |
| Deep Learning 4j | 0.999 | 0.015 | 0.985 | 0.001 | 0.999 | 0.999 | 0.999 | 0.996 |
| J.48 | 0.999 | 0.024 | 0.976 | 0.001 | 0.998 | 0.999 | 0.999 | 0.989 |

TABLE IV: Experiment 2: New test-set

| Performance Evaluation on Dataset 1 using unseen dataset | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ML Algorithm | TPR | FPR | TNR | FNR | Accuracy | F1 Measure | Precision | Area Under ROC (AUC) |
| RBF | 0.999 | 0.129 | 0.871 | 0.001 | 0.994 | 0.997 | 0.995 | 0.978 |
| Random Forest | 0.998 | 0.010 | 0.990 | 0.002 | 0.998 | 0.999 | 1.000 | 1.000 |
| MLP | 0.998 | 0.030 | 0.970 | 0.002 | 0.997 | 0.999 | 0.999 | 1.000 |
| Deep Learning 4j | 0.960 | 0.005 | 0.995 | 0.040 | 0.965 | 0.979 | 0.999 | 1.000 |
| J.48 | 0.998 | 0.030 | 0.970 | 0.002 | 0.996 | 0.998 | 0.999 | 0.995 |

parameters. The concept drift detector analyzes a labeled test dataset after a certain interval of time. The test dataset is based on latest malicious samples from different malware families, selected by the security expert. The dataset will be tested to analyze the efficiency of the classifier. If the accuracy of the classifier is below the threshold for any particular malware family, the detector will trigger the retraining phase after including the instances of that family to the training dataset. These testing samples will be collected and labeled by the security expert, so it will be difficult or nearly impossible for an adversary to poison our concept-drift system by forcing the learning algorithm to learn the patterns suitable for her. The full working principle and architecture of this concept-drift detector will be discussed in our future work.

## IV. Performance Evaluation

The following parameters were used in order to evaluate ML Classifiers.

TABLE V: Confusion Matrix

| | | Predicted | |
|---|---|---|---|
| | | Malicious | Normal |
| **Actual** | Malicious | $TruePositive$ | $FalseNegative$ |
| | Normal | $FalsePositive$ | $TrueNegative$ |

True Positive (TP): Malicious instance classified as Malicious.
False Positive (FP): Normal instance classified as Malicious
False Negative (FN): Malicious instance classified as Normal.
True Negative (TN): Normal Instance classified as Normal.

$$TPR = \frac{TP}{TP+FN}$$

$$FPR = \frac{FP}{FP+TN}$$

$$TNR = \frac{TN}{TN+FP}$$

$$FNR = \frac{FN}{TP+FN}$$

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

$$Precision = \frac{TP}{TP+FP}$$

$$F1Measure = \frac{2*(TPR*Precision)}{TPR+Precision} = \frac{2TP}{2TP+FP+FN}$$

ROC (Receiver Operating Characteristic) curve is a plot between TPR and FPR at various threshold settings [31]. The area under ROC Curve (AUC) used in this study is derived from ROC Curve. AUC tells how efficient will be the classifier on the unseen data. AUC value is the best parameter to compare different algorithms. Furthermore, accuracy depends on both TP and FP and therefore considered as an important parameter when it comes to ML classifiers performance evaluation.

## V. Experiments

We have performed two experiments using 5 ML algorithms. These experiments were performed in WEKA [32]. The tuning of the hyper-parameters of the ML algorithms was also performed using WEKA. In the first experiment, the cross-validation [33] method was used for the performance evaluation. The cross-validation (CV) is very famous method when evaluating the machine learning algorithms. In this method, the validation set is not needed, while the test-set is held out for final evaluation. We have used 10 fold CV, it divides the training set into 10 smaller subsets. The optimizing iteration goes for 10 times and in each iteration, 9 data subsets were used as training set and 1 remaining subset used for testing and evaluating the accuracy of the ML algorithm. The overall performance of the classifier will be the average of values obtained in each iteration. Although CV method is computationally expensive, it avoids the chance of over-fitting the model. Furthermore, in the second experiment, the classifier was tested on the unseen data. To avoid the issue of over-fitting, the dataset was divided into three portions: the training set, validation set and test set. The training was done on the training dataset and then evaluation was performed on

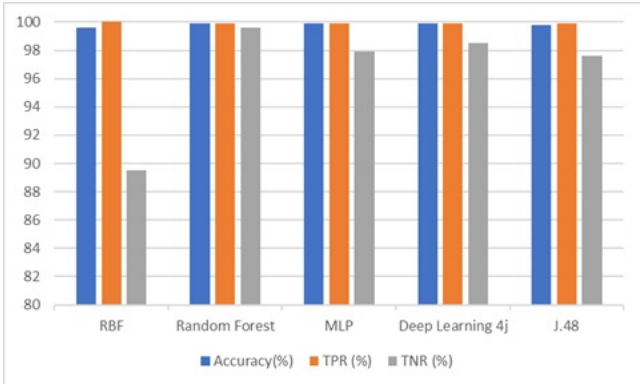


Figure 2: Performance evaluation: Cross validation

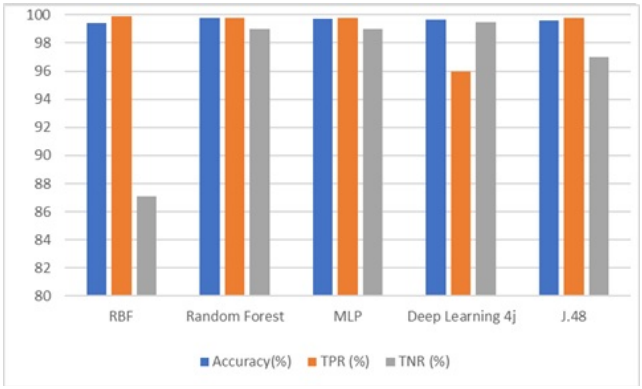


Figure 3: Performance evaluation: New Test Dataset

the validation set for optimization. Finally, the evaluation was performed on the test set.

In the first experiment, DeepLearning4j and Random Forest produced the highest accuracy of 99.9%, while the RBF produced the worst performance with FPR of 10.5%, as shown in the Table III. In the Fig. 2, the best TPR and TNR were observed by Random Forest and DeepLearning4j. In the second experiment, the training dataset and test dataset were different. The Random forest, MLP and Deeplearning4j produced best results with an accuracy of 99.9% as shown in the Table IV. It can be seen in Fig. 3, RBF produced worst performance among all as it produced high FPR. The best performance in both experiments was observed by Random Forest, MLP and Deeplearning4j.

## VI. Conclusion

The threat detection model developed in this study can detect known and unknown threat by using time-based features. ML-based Intrusion detection systems need frequent retraining. The framework proposed in this paper automates the retraining phase when the accuracy goes below the threshold. The retraining-phase overcomes many limitations in ML-based Intrusion detection systems that produce false positives or false negatives after an interval of time. In our previous studies, we have observed that ensemble of ML algorithms could be used to further enhance the performance of the classifiers. Proper feature extraction plays a vital role in ML-based Intrusion detection systems. Wrong features could overfit the model and produce less accuracy on unseen traffic. The classifiers built in this study were able to provide the accuracy of 99.9%, which is higher than many of the other studies in this field. Future work in progress is to build a model which contains several other malware families and list of other benign applications. The concept drift detector will be further enhanced to behave towards the changing attack patterns.

## References

[1] Statista, "Mobile phone users worldwide 2013-2019," Tech. Rep., Access Date 30 Nov, 2017. [Online]. Available: https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide

[2] "Smartphone os market share, 2017 q1," IDC, Tech. Rep., Access Date 30 Nov, 2017. [Online]. Available: https://www.idc.com/promo/smartphone-market-share/os

[3] McAfee, "Trojans, ghosts, and more mean bumps ahead for mobile and connected things (what lies ahead for 2017)," Tech. Rep., 2017, Access Date 30 Nov, 2017. [Online]. Available: https://www.mcafee.com/us/resources/reports/rp-mobile-threat-report-2017.pdf

[4] K. Timm, "Strategies to reduce false positives and false negatives in nids," Tech. Rep., 2001, Access Date 10 Sep, 2017. [Online]. Available: http://www.symantec.com/connect/articles/strategies-reduce-false-positives-and-false-negatives-nids

[5] K. Julisch and M. Dacier, "Mining intrusion detection alarms for actionable knowledge," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 02*. Association for Computing Machinery (ACM), 2002.

[6] R. Koch, "Towards next-generation intrusion detection," in *Cyber Conflict (ICCC), 2011 3rd International Conference on*. IEEE, 2011, pp. 1–18.

[7] J. A. Copeland III, "Flow-based detection of network intrusions," Feb. 27 2007, uS Patent 7,185,368.

[8] "Malware detection and subscriber protection infographic," Nokia, Tech. Rep., Access Date 30 Nov, 2017. [Online]. Available: https://networks.nokia.com/solutions/security-guardian-infographic

[9] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," *Soft Comput*, nov 2014.

[10] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket," in *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS)*, 2014.

[11] A. Feizollah, N. B. Anuar, R. Salleh, F. Amalina, R. R. Maarof, and S. Shamshirband, "A study of machine learning classifiers for anomaly-based mobile botnet detection," *Malaysian Journal of Computer Science*, vol. 26, no. 4, 2014.

[12] X. Su, M. C. Chuah, and G. Tan, "Smartphone dual defense protection framework: Detecting malicious applications in android markets," in *Mobile Ad-hoc and Sensor Networks (MSN), 2012 Eighth International Conference on*. IEEE, 2012, pp. 153–160.

[13] S. Kumar, A. Viinikainen, and T. Hamalainen, "Machine learning classification model for network based intrusion detection system," in *Proc. 11th Int. Conf. for Internet Technology and Secured Transactions (ICITST)*, Dec. 2016, pp. 242–249.

[14] S. Wang, Z. Chen, L. Zhang, Q. Yan, B. Yang, L. Peng, and Z. Jia, "Trafficav: An effective and explainable detection of mobile malware behavior using network traffic," in *Quality of Service (IWQoS), 2016 IEEE/ACM 24th International Symposium on*. IEEE, 2016, pp. 1–6.

[15] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 95–109.

[16] D. Zhao, I. Traore, B. Sayed, W. Lu, S. Saad, A. Ghorbani, and D. Garant, "Botnet detection based on traffic behavior analysis and flow intervals," *Computers & Security*, vol. 39, pp. 2–16, 2013.

[17] M. Stevanovic and J. M. Pedersen, "An efficient flow-based botnet detection using supervised machine learning," in *Computing, Networking and Communications (ICNC), 2014 International Conference on*. IEEE, 2014, pp. 797–801.

[18] C. Livadas, R. Walsh, D. Lapsley, and W. T. Strayer, "Usilng machine learning technliques to identify botnet traffic," in *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*. IEEE, 2006, pp. 967–974.

[19] S. S. Kowsalya, "Website fingerprinting using traffic analysis attacks."

[20] A. Hintz, "Fingerprinting websites using traffic analysis," in *International Workshop on Privacy Enhancing Technologies*. Springer, 2002, pp. 171–178.

[21] Virustotal.com. [Online]. Available: https://www.virustotal.com/

[22] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. v. d. Veen, and C. Platzer, "Andrubis – 1,000,000 Apps later: A view on current android malware behaviors," in *Proc. Third Int. Workshop Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, Sep. 2014, pp. 3–17.

[23] I. M. Digit Oktavianto, *Cuckoo Malware Analysis*. Packt Publishing, 2013.

[24] B. H. Trammell and E. Boschi, "Bidirectional flow export using ip flow information export (ipfix) : Rfc-5103," IETF, Tech. Rep., Access Date 01 Jan, 2015. [Online]. Available: https://tools.ietf.org/html/rfc5103.html

[25] E. Boschi and B. Trammell, "Bidirectional flow measurement, ipfix, and security analysis," 2006.

[26] S. Kumar, A. Viinikainen, and T. Hamalainen, "Evaluation of ensemble machine learning methods in mobile threat detection," in *Proc. 12th Int. Conf. for Internet Technology and Secured Transactions (ICITST) , in press.*, Dec. 2017.

[27] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. ACM, 2006, pp. 16–25.

[28] M. Kloft and P. Laskov, "Security analysis of online centroid anomaly detection," *Journal of Machine Learning Research*, vol. 13, no. Dec, pp. 3681–3724, 2012.

[29] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. ACM, 2011, pp. 43–58.

[30] J. Newsome, B. Karp, and D. Song, "Paragraph: Thwarting signature learning by training maliciously," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2006, pp. 81–105.

[31] F. J. Provost, T. Fawcett, and R. Kohavi, "The case against accuracy estimation for comparing induction algorithms." in *ICML*, vol. 98, 1998, pp. 445–453.

[32] R. Quinlan, "4.5: Programs for machine learning morgan kaufmann publishers inc," *San Francisco, USA*, 1993.

[33] Cross-validation: evaluating estimator performance. Access Date 30 Nov, 2017. [Online]. Available: http://scikit-learn.org/stable/modules/cross_validation.html