A Lightweight Network-based Android Malware Detection System

Igor Jochem Sanz*, Martin Andreoni Lopez*,
Eduardo Kugler Viegas*†, and Vinicius Rodrigues Sanches*

*Samsung Eletrônica da Amazônia - SRBR, Campinas, SP, Brazil
†PPGIa - Pontifícia Universidade Católica do Paraná, Curitiba, PR, Brazil**
{igor.js, m.andreoni, v.sanches}@samsung.com, eduardo.viegas@ppgia.pucpr.br

***Abstract*—Over the last years, mobile devices became target of thousands of malicious applications. Since then, several works have proposed and evaluated highly accurate machine-learning malware detection schemes. However, these schemes are hardly used in production, either because of their resource-intensive nature for deployment in mobile devices or due to high false alarm rates. This paper proposes a lightweight malware detection system by means of network behavior analysis. Our system relies on lightweight machine-learning techniques to monitor network behavior of suspicious applications. To evaluate our proposal, we construct a realistic and up-to-date network traffic dataset made of 359 goodware and malware applications. The evaluation results show that our proposal is able to detect new malware variants with accuracy near 90% and false-positive rates below 3% using only 14 features inferred directly from the TCP/IP packet header. In addition, when deployed in a Samsung Galaxy S9+, our technique consumes on average less than 5% of CPU, even in network peaks of 90 Mb/s.**

I. INTRODUCTION

Mobile devices became ubiquitous in the current super-connected world. The number of smartphones is growing each year and is expected to reach six billion marks by 2020 [1]. The mobile popularity is also gaining prominence in network security, as it paves the way to the development of malicious software targeting those devices. Malicious mobile applications (malware) have already surpassed the 25 million mark in 2018 [2], with the majority tailored to Android operating systems [3], as it comprises 70% of mobile devices [4]. Android malware applications (APK) are often used by hackers to gather user sensitive information or even remotely control user devices for profit purposes. In general, Android malware tries to counterfeit a version of benign applications of the official Google Play market or unofficial APK markets, such as APKMirror, APToide or GetAPK, making their detection more challenging.

Many approaches have been proposed to characterize mobile application traffic [5] and to detect malicious applications on mobile devices [6]. The literature divides mobile malware detection techniques into two approaches: static and dynamic analysis [7]. The static approach comprises passive techniques that explore the applications without executing its code. Such analysis includes examining the source or binary code, or even evaluating the APK requested permissions. Sophisticated malware, however, uses code obfuscation techniques to avoid static analysis detection. On the other hand, the dynamic analysis relies on monitoring malware behavior by collecting information during the application execution to examine its behavior, such as system calls, accessed files or network traffic [8], [9]. To analyze malware, researchers often perform dynamic analysis in a simulated or restricted environment, where sophisticated malware can hide their malicious behavior to prevent being detected [10]. Furthermore, most dynamic analysis techniques are not suited for mobile devices. This is because, in general, such techniques are executed in simulated environments that do not take into account the mobile device resource restrained nature. Besides, most malicious applications use the network to connect with a remote server, download malicious code or even communicate with a botnet. Identifying network-based fingerprints of security threats is challenging since it requires the generation of a dataset of network behavior triggered by real user actions and by malicious activities during application execution [11].

In light of this, this paper proposes a lightweight network-based malware detection system to detect malicious Android applications at run time. The proposed system relies on lightweight machine-learning techniques that run on the top of Android smartphones without significantly affecting its performance. We build a realistic dataset that performs interactions with applications by capturing on-device malware and goodware APK network usage during a period of time. Then, a lightweight feature extractor based on static ring buffer and tailored for Android devices abstracts network traffic into numerical features inferred directly from TCP/IP packet header. Finally, two lightweight machine-learning algorithms, AdaBoost and Random Forest, applied directly to the ring buffer output, real-time classify the APK traffic behavior. To reduce false alarms to the user, we create a network malware score that represents the fraction of packets classified as malicious during APK execution. The malware score assesses the APK network behavior risk, and enable to adjust the trade-off between false-positive and false-negative rates according to the number of malicious packets. Finally, we develop a

** The author changed affiliation to PUC-PR during paper submission process.

Annex to ISBN 978-3-903176-28-7 ©2020 IFIP

prototype of our system that achieves high accuracy in detecting malicious applications without being resource-intensive. Therefore, the contributions of this paper are fourfold:

- A lightweight network-based malware detection system suited for energy-restrained devices with near 90% of accuracy and low false alarm rates and processing demands. Our detection scheme addresses the challenges of embedding a network packet classification architecture in energy-restrained devices, such as mobile devices. With only 14 TCP/IP packet features, our proposal is capable of classifying correctly more than 90% of malicious traffic with a false positive rate below 3%, without deeply inspecting packets;
- A realistic and up-to-date dataset of Android malware and goodware network behavior comprising 3.36 GB of stimulated network traffic obtained from 359 Android applications. The dataset is the first of its kind to specifically monitor Android APKs network behavior in real mobile devices, without being in a simulated or restricted environment;
- A fundamental analysis of the capacity of our proposal detecting malicious traffic from different mobile malware families and malware variants previously unseen by the classifier.
- A prototype of our system built on top of a Samsung Galaxy S9+ showing the feasibility and practical results achieved by our proposal. We observed that, even in peaks of 90 Mb, the prototype consumes in average 5% CPU resources. Moreover, as we only use a small period of time of APK network traffic to provide a classification result, CPU resources are only demanded at the beginning of the application, maintaining near 0% of resource consumption when the proposal is in idle state.

The remainder of this paper is organized as follows. Section II presents the lightweight network-based malware detection system and its prototype. Section III details the construction of our dataset. Section IV evaluates our proposal. Section V discusses the limitations of our work and correspondent solutions. Section VI discusses related work. Finally, Section VII concludes the work.

II. A LIGHTWEIGHT NETWORK-BASED MALWARE DETECTION SYSTEM

We aim to detect malware APKs executed on mobile devices according to their network behavior over a period of time. Our premise is that malicious network behavior remains similar even if attackers modify malware signatures and system call behavior. Therefore, since the analysis of network packets is often a processing consuming task, a lightweight detection system must be designed to enable malware classification in real mobile devices.

Figure 1 shows the design of the proposed lightweight network-based malware detection system, which relies on machine learning to distinguish malware and goodware APKs according to their network behavior over time. As the system runs in Android devices, the proposed classification scheme

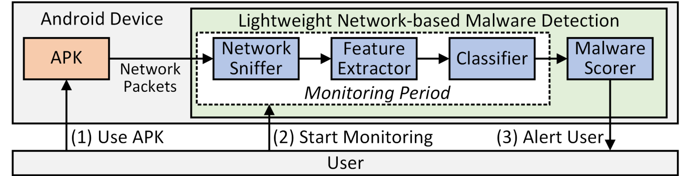


Figure 1: The Design of the Proposed Lightweight Network-based Malware Detection System.

must consider the demanded processing to fulfill such a task. Hence, we analyze the perspective of a mobile device user who wishes to monitor an APK in the mobile device (1). Then, the monitored APK runs for a given period of time (2) while having its generated network packets monitored. Flow-based features, extracted over time and assigned to each generated network packet, represents how the APK communicates with remote servers at the application level. Thus, a machine learning model classifies each feature vector as either goodware or malware. As the system classifies network packets individually, a threshold determining if the APK is malicious or not must be defined. Thus, our proposed system relies on the *Anomaly Score* computation, which measures the fraction of network packets classified as malicious to all generated packets (3). The *Anomaly Score* aims to classify an APK as malware only if the network data over time has a fraction of malicious packets greater than a defined threshold.

In general, network-based feature extractors rely on dynamic memory allocation to extract statistical features from the network traffic, without taking into account the energy-constrained nature of mobile devices. We use an optimized lightweight feature extractor based on static memory allocation tailored to execute on Android-based devices [12]. Our proposal extracts numerical features from the TCP/IP packet header in a flow-based approach, while composes a feature vector for each network packet. Numerical features allow to decrease the processing of feature extraction since they are simple windowed counters of information that are quickly inferred from TCP/IP packet header. We are also able to decrease the demanded processing for the feature extraction task since we perform classification without the storage of all network packets occurred during a period of time. A packet-granularity classification must consider two aspects that may compromise a lightweight system. First, as we classify each generated network packet, many false alarms might trigger during the monitoring period. Second, as we generate more classification events, an inadequate model selection may incur in additional processing. Hence, our system addresses both aspects in a twofold manner: using an *Anomaly Score* threshold and a lightweight classifier. The *Anomaly Score* raises an alarm only if a significant portion of network packets belonging to the monitored APK is classified as malware. To reduce the processing demanded the classification task, we select two tree-based classifiers suitable for non-linear data, which are known to scale and have a good trade-off between accuracy and classification processing time [13].

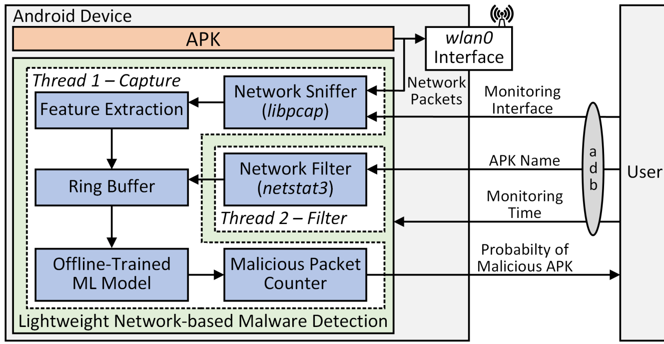


Figure 2: Prototype Architecture of the Proposed Lightweight Network-based Malware Detection System.

A. Anomaly Score

Our proposed system classifies malware in a network packet granularity to provide a lightweight detection approach. Multiple packet instances belonging to the same APK need to be classified, nevertheless, not all of them may be labeled to the same class. Therefore, the *Anomaly Score* module establishes whether a monitored APK is a malware or not according to the classified events over a period of time and adjusts the ratio of false-positives and false-negatives during an APK monitoring. Equation 1 defines the *Anomaly Score*, where x is the evaluated network packet outcome and N is the total number of evaluated network packets of the specific APK.

$$Anomaly\ Score = \frac{\sum_{i=0}^{N} f(x) = \begin{cases} 1, & \text{if } x = malware \\ 0, & \text{otherwise} \end{cases}}{N} \quad (1)$$

The computation of *Anomaly Score* allows defining a desired alert threshold. In our scenario, we aim to reduce the number of false alerts to the user. The reason is simple, a high number of false alerts will impact the user experience and trust onto our system and user will be more tempted to discredit and ignore critical alerts. Then, we set a high *Anomaly Score* threshold (evaluated in Section IV). Nonetheless, if we change objective to minimize false-negatives, we would adjust our anomaly score threshold accordingly, such that no malware APK passes unnoticed.

B. Prototype Architecture

The architecture of our implemented system is shown in Figure 2. The system receives three parameters, the Monitoring Interface name, the APK name and the Monitoring Time. The Monitoring Interface defines the network interface to be monitored (e.g. *wlan0*). The APK name defines from which APK the packets must be monitored. Finally, a decision regarding the APK class (e.g. goodware or malware) occurs only after the Monitoring Time finishes. Thus, the prototype contains two main threads, namely Capture Thread and Filter Thread.

The Capture Thread handles the network packet monitoring, feature extraction, and classification tasks. This thread is composed of the following modules: network sniffer, feature extraction, ring buffer, classifier, and malicious packet counter. First, the network sniffer reads the network packets from the Monitoring Interface, using the *libpcap*[1] API. Afterward, the feature extraction module extracts 21 numerical network packet features. The features are packet frame length (1), number of frames and number of bytes, from source to destination and from destination to source (4), number of frames and number of bytes with the same port for both directions (4), and number of packets with PSH, SYN, FIN, ACK, SYN, and RST flags active for both directions (12). Thereby, the feature set comprises information gathered directly from the packet header, such as TCP header flags, connection status, and flow-based information. In such context, flow-based information comprises data about the number of packets and bytes in forward and backward direction over the last four seconds. The four-second flow time gathering was established empirically. Then, the extracted feature vector is stored in the ring buffer module, which holds the last *n* read samples. The ring buffer module stores the extracted feature vectors until the APK that originated them is known.

In contrast, the Filter Thread comprises the Network Filter module, which periodically executes the *netstat3*[2] application. The Network Filter module defines which network sockets were opened by the monitored APK. Therefore, by periodically using such information, the ring buffer module is able to filter the feature vectors according to the APK that generated the network packets. Thus, once the output of the network statistics module is presented, the ring buffer module drops the samples that unmatched the monitored application flows. The Network Filter module periodically executes the ring buffer cleanup in a one-second interval.

Finally, for the classification task, an offline-trained Machine Learning model is loaded in the system memory. Then, the samples filtered by the Network Filter module are sent to the classifier. The classifier predicts the sample values as a binary classification either goodware or malware. After the Monitoring Time, the Malware Scorer module establishes whether the monitored APK is malware or not.

III. Testbed and Dataset Creation

Sophisticated Android malware is capable to detect if it is being executed in a sandbox environment and to disguise malicious behavior during the sandbox execution. Also, malware malicious actions may only be activated when specific events are triggered on the device, e.g., a user touching a specific button. Hence, the creation of a realistic Android malware testbed is a difficult task. As a solution, we create a realistic dataset by capturing pcap files of real network traffic of applications executed in a smartphone device, avoiding simulation and preventing malware disguising techniques. We aim to improve test coverage and scalability as much as possible rather than focus on most cases of normal use, then,

[1] Libpcap: https://www.tcpdump.org/
[2] Netstat: https://github.com/LipiLee/netstat

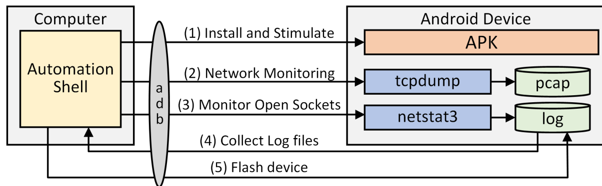


Figure 3: Testbed automation scheme for Android APKs network monitoring. Each APK is monitored individually through five steps for dataset generation.

we use a tool to stress the installed application and create a pseudo-random stream of user events into the smartphone.

We built an Android malware testbed using the procedure shown in Figure 3. The testbed must enable malware and goodware network usage monitoring in realistic settings for a long period. To fulfill such requirements and scale, we built an automation process in which each monitored APK undergo the following procedures: i) *Install and Stimulate*. Using Android Debug Bridge (ADB) tool[3], the APK to be monitored is installed on a real Android device. After the installation, the APK is stimulated using ADB monkey command, which sends several pseudo-random events to the APK. Therefore, all APKs can be installed and stimulated in real Android devices; ii) *Monitor Network*. In parallel to the APK stimulation stage, the network interface (e.g. *wlan0*) is monitored using *tcpdump*[4] tool. The tool generates a PCAP file with all network packets sent and received by the network interface; iii) *Monitor Open Sockets*. Also, in parallel to APK stimulation stage, the monitored APK opened network sockets are logged to a file using *netstat3* tool. This log file is used for filtering the PCAP file after the monitoring stage, therefore we can establish which network packets were sent or received by the monitored APK; iv) *Get log files*. After a given time window, the APK stimulation ends and the PCAP and log files are sent back to the computer; v) *Flash Device*. Before starting a new APK monitoring task, the device is fully flashed to remove any possible interference between each APK monitoring. By automating the above procedure, we can realistically monitor APK network usage.

Two groups of APK were assembled in our testbed: goodware and malware. For the goodware group, we developed a script to download real APKs from the Google Play Store using the *gplayclitool*[5]. Google Play Store uses Google Play Protect and therefore, the downloaded APKs are deemed as goodware. We select and download the 500 most popular APKs from Google Play Store for the goodware group. On the other hand, for the malware group, samples from Android Malware Dataset [14] were used as our baseline dataset. Android Malware Dataset (AMD) comprises 24,553 malware samples divided into 71 malware families. In a preliminary analysis, we restrict our malware scope to the families that are known to generate malicious network traffic. Each selected APK sample, from both groups, was executed for a period of time using the five-step procedure. We follow the malware traffic characterization performed by Chen *et al.*, which used a capture time of 500 seconds [15]. By this time of the process, we captured 7.17 GB of network data from 787 APKs, in which 395 are goodware and 392 are malware. Because our dynamic analysis is focused on network malicious traffic, the next step is filtering samples that did not generate enough network traffic by a defined threshold. We define 500 packets as the minimum to include the APK sample in the final dataset. This is necessary to provide substantial data to train the machine-learning models. After the filtering process, the final dataset comprises 3.36 GB of network data from 359 APKs among 13 malware families. Table I summarizes the network traffic distribution of the final dataset.

[3]ADB: https://developer.android.com/studio/command-line/adb
[4]Tcpdump: https://www.tcpdump.org/
[5]GPlayCli: https://github.com/matlink/gplaycli

Table I: Network data used for the experiments according to each type of selected APK after filtering APKs that generated less than 500 packets.

Family	# APKs	Size	# Packets
Goodware	266	376 M	620437
FakeAngry	2	467 K	3674
FakeAV	3	4.32 M	13530
FakeDoc	6	1.41 M	5223
GingerMaster	6	10.90 M	15485
GoldDream	3	5.34 M	9695
Ksapp	6	401 K	5370
MMarketplay	3	3.26 M	20359
MobileTX	12	6.89 M	32387
Mseg	33	3.29 G	3629321
Mtk	8	1.64 M	9262
Nandrobox	3	437 K	5188
Vmvol	6	33.89 M	60031
Winge	2	719 K	3886
Total	359	3.36 G	3813411

IV. Evaluation and Discussion

Our evaluation aims to answer three research questions: i) *Does network traffic from malware and goodware differ significantly to enable APK classification?*; ii) *Is it possible to detect unknown malware variants by the means of network traffic classification?*; and iii) *What is the processing trade-off of network traffic classification in Android devices*?

A. Model Building

As previously discussed, our prototype loads in-memory machine learning models built offline. The model building is a processing consuming task, which demands dedicated hardware to execute. Hence, we define as lightweight, models that classify fast enough to be done at runtime, regardless of the processing cost of the offline training. As the proposed system is agnostic of the learning technique employed, our goal is not to select or develop the lightest machine-learning models. Instead, our goal is to evaluate the feasibility of achieving lightweight detection with machine-learning models deployed in real-devices, and on near real-world conditions. Then, we select two classifiers that have a good trade-off between accuracy and classification time, Random Forest (RF) and AdaBoost. These classifiers are also adopted by researchers because of their *if-then-else* format and their efficiency for packet classification and intrusion detection, making them suitable for resource-constrained devices [16].

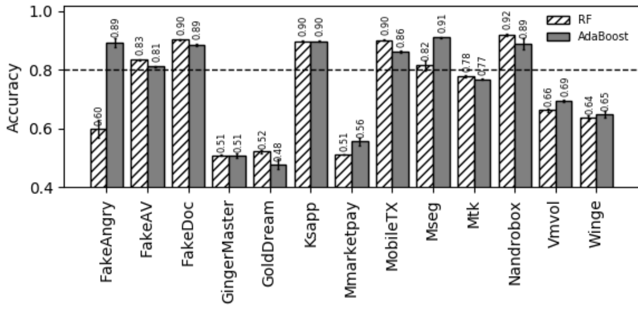
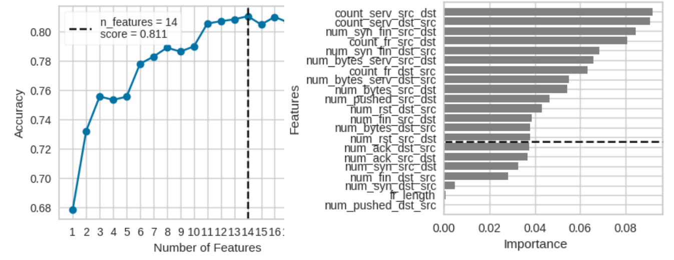


Figure 4: Accuracy of packet classification of the Random Forest and AdaBoost classifier trained with each malware family separately. The dashed line indicates a 80% accuracy threshold.



(a) 0.81 of accuracy is achieved with 14 features.

(b) Feature Importance.

Figure 5: Recursive Feature Elimination using Random Forest model with 10-fold cross validation. The dash line indicates that 14 features are enough to get 81% of accuracy of packet classification.

To answer question i), both classifiers were trained multiple times using an one-vs-one approach, each time individually for a different malware family. This evaluation step aims to establish which malware families are feasible to be detected using network traffic features, since some existent malware families simply does not generate malicious traffic. The one-vs-one training produced 26 model generation processes. The classifiers were built using *scikit-learn*[6] v0.20.1 tool. In addition, to assess the capability of detecting malware variants previously unseen by the classifier, we exclusively split the APKs of the same family between training and test groups. In other words, we guarantee the same APK network traffic packets is not present in both training and test sets simultaneously.

Since benign packets are more voluminous than malicious packets (Table I), we balance the dataset using random under-sampling for the majority class. Moreover, the train and test dataset were split considering the number of APKs of each malware family. Thus, we split approximately 75% of APKs of a given family for training and the remaining 25% for testing.

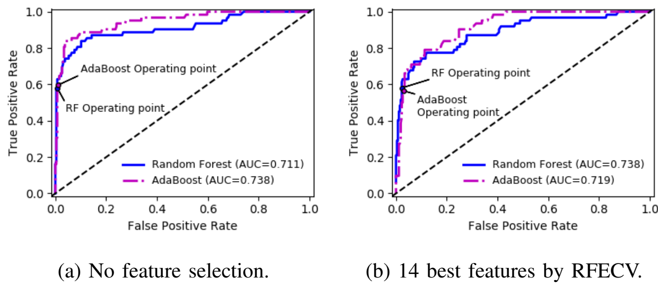
Figure 4 shows packet classification accuracy using all features with Random Forest and AdaBoost models when we train them separately by malware families, using the one-vs-one approach. For each one of the 26 family-trained models, we repeated model generation process 20 times varying the random seed of the under-sampling each iteration, and we show results with a confidence interval of 95%. This experiment presented an accuracy above 80% for 6 and 7 families with Random Forest and AdaBoost respectively, out of 13 malware families. We observed that for six malware families on both classifiers, the accuracy was above 80% with a false positive rate below 5%. The best six families were FakeAV, FakeDoC, Ksapp, MobileTX, Mseg, and Nandrobox.

Concerning feature selection, Figure 5 shows results obtained using Recursive Feature Elimination with Cross-Validation (RFECV). In RFECV, the best feature subset is searched using a greedy optimization approach. We use 10-fold cross-validation for each subset of features and the Random Forest algorithm to evaluate the feature subsets.

[6]Scikit-learn: https://scikit-learn.org/

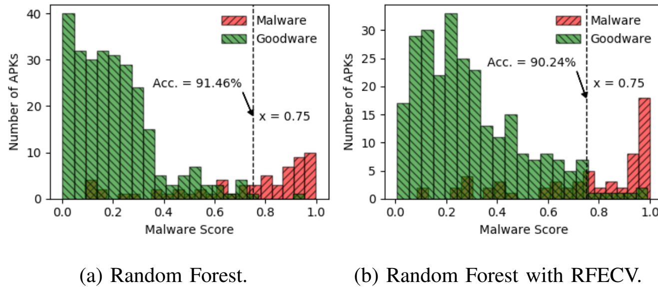
Figure 5a presents the results of the RFECV. We observe that only 14 features out of 21 are enough to achieve 81% of accuracy for malicious packets classification. Figure 5b illustrates the feature importance given by the RFECV method. The dashed line separates the 14 most important features selected by the algorithm. These results support that 6 out of 13 malware families can be detected by the means of network traffic behavior analysis. We conclude our finding for research question i) that goodware and malware does differ significantly for specific classes of malware, while for others it does not significantly present malware traffic behavior to deviate from goodware behavior. Since our proposal relies on network behavior analysis, we henceforth restrict our final model to the 6 best families.

The Random Forest and AdaBoost algorithms contain different parameters to be tuned and finding the optimal solution in an exhaustive task. Thus, as a last step in the model building process, we aim to tune the hyper-parameter values to the dimensions of our experiment, i.e., to the number of instances, number of features, class distribution, and feature importance. We use a random grid search algorithm to select the best set of parameters of $number\ of\ trees$ and $maximum\ depth$ for Random Forest and $number\ of\ trees$ and $learning\ rate$ for AdaBoost. We perform three-fold cross validation with 4000 total combinations with 100 random iterations in an Octa-core 2.63 GHz Xeon 5600 Server, with 32 GB RAM, running on Ubuntu bionic 18.04.1. The three most relevant cases that presented the best-balanced accuracy results were achieved with a $maximum\ depth$ of 10, and $number\ of\ trees$ varying between 10 and 30, which we select 30. Similarly, for the AdaBoost model, the $number\ of\ trees$ was selected as 50 and the $learning\ rate$ as 1. Since hyper-parameter optimization is responsible for only a small fraction of the final model accuracy, after we tailor parameters to address our classification problem, it is not necessary to repeat this process every model training, unless a major experiment change occurs, such as a concept-drift on the dataset. We emphasize the results achieved in this step are not generalizable, and this procedure should be independently executed for other experiment conditions. Moreover, the entire model building process occurs on the cloud and only after the final model is generated, i.e., the



(a) No feature selection. (b) 14 best features by RFECV.

Figure 6: ROC curve for Random Forest and AdaBoost classifiers, as we vary the malware score threshold for both scenarios without feature selection and using only the 14 best features selected by RFECV technique.



(a) Random Forest. (b) Random Forest with RFECV.

Figure 7: Histogram of goodware and malware APKs separated by different ranges of malware scores for the Random Forest algorithm.

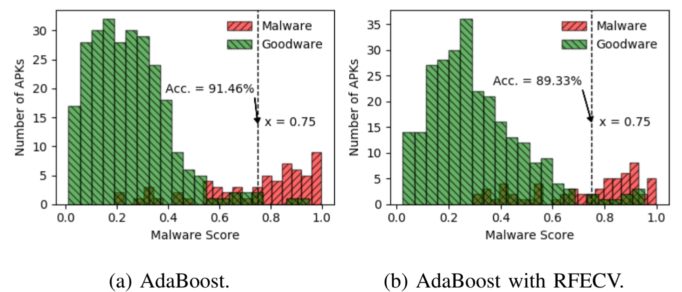
parameters and the tree-based structures, model is ported to the smartphone.

B. Final model evaluation and trade-offs

The model-building phase consisted of evaluating the classifier performance at the granularity of packet classification. The final model to the user, however, should return the APK class. Therefore, we evaluate the performance of APK classification for the entire dataset to answer question ii).

Figure 6 presents the receiver operating characteristic (ROC) curve for both algorithms in the APK classification for the entire dataset as we vary the Malware Score threshold. For both classifiers, we compare two feature sets, all 21 numerical features and the best 14 features selected through RFECV technique. With no feature selection, AdaBoost showed a better Area Under Curve (AUC), indicating an overall better classification performance. When RFECV feature selection is applied, instead, Random Forest achieves similar AUC (0.738) but with fewer features to process. As discussed on Section II-A, we aim to reduce the number of false malware alerts. Figure 6 shows that *Anomaly Score* threshold near 0.75 achieves the best trade-off between false-positive and true-positive for all scenarios evaluated, according to our classification goal. Then, we select an operating point for a 0.75 malware score threshold, i.e., we classify the APK as malicious if 75% of packets are classified as malicious during the monitoring period.

Figure 7 and Figure 8 depict the separation between malware and goodware obtained in both classifiers using the selected operating point (OP). We note that only a small



(a) AdaBoost. (b) AdaBoost with RFECV.

Figure 8: Histogram of goodware and malware APKs separated by different ranges of malware scores for the AdaBoost algorithm.

portion of goodware APKs was misclassified as malware. On the selected OP, the best accuracy for APK classification was 91.46%, for both AdaBoost and Random Forest algorithms when no feature selection is applied. By selecting the best 14 features, however, the accuracy remained similar, 90.24% for the Random Forest and 89.33% for AdaBoost. For all scenarios evaluated, the false positive rate of APK classification remained under 3%, and, specifically for the Random Forest with no feature selection, this rate was 0.07%. We highlight that such results were achieved using one general machine-learning model for all selected families. A unique model to detect multiple malware families and their variants with few false alarms, instead of multiple models tailored per family, is a desirable property for lightweight detection.

The aforementioned results of our system show that detecting malware variants by means of network features is feasible in terms of classification performance. While the classification performance per packet achieves around 80% accuracy, when we apply the *Anomaly Score*, we are able to classify an APK between malware or goodware with near 90% of accuracy. The machine-learning algorithms were able to recognize patterns on the network behavior that distinguish well malware applications from goodware applications. Literature has shown that many Android applications did not implement adequate standards for secure network communications and precarious implementations are prevalent in malware APK [17]. Moreover, malware variants from the same malware family shares similar part of codes and communication patterns, which are possible to be detected through network analysis [18]. Thus, we shown our system was able to detect high-level characteristics of malware APK using a combination of features such as the number of bytes, number of each flags of TCP, number of frames, frame length, and number of bytes transmitted in both directions in a certain period of time. Moreover, the resultant feature set is lightweight to extract because it can be inferred directly from header of TCP/IP packets, with almost no feature processing, and demonstrates that a lightweight network-based malware detection system is feasible for resource constrained-devices. Moreover, as we split APK between train and test sets in a way that data from the same APK is not present in both sets simultaneously for the same iteration, the system was capable of detect variant of malware previously unseen.

Table II: Performance analysis of the prototype implementation monitoring two different applications.

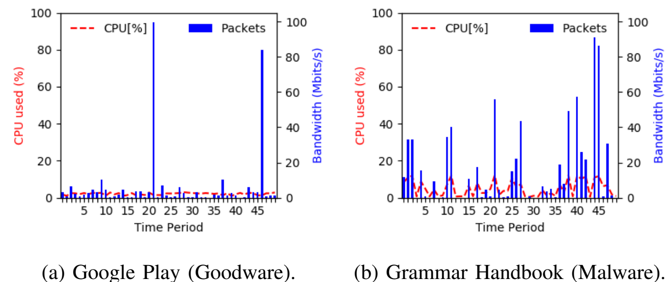
APK	Type	Mal. Score	# Captured Pkt.	# APK Pkt.	# Susp. Pkt.	Avg. CPU Used[%]
Grammar handbook	Malware	83.06%	4182	1447	1202	8.82
Google Play	Goodware	0%	2489	2440	0	4.14

C. Performance Evaluation

To answer question iii), we implemented our prototype in a Samsung Galaxy S9+ with the Qualcomm SDM845 Snapdragon 845 chipset and 6GB of RAM. The only practical requirement for the prototype runs and extracts features from wireless card data is to use rooted devices, although this would not be required for a final deployment embedded natively in the Android. We use SimplePerf[7], a native profiling tool for Android that reports statistics about CPU performance and OS resource usage, to gather statistics of our prototype implementation. Thus, we measure only the resources consumed by our malware detection system, and not by the monitored APK.

During a network traffic peak, our prototype must prove resilience to high rate of packet capture, feature extraction, filtering, and classification steps simultaneously. Therefore, we analyze prototype not only at normal execution, but under the stress condition of the peak of network processing. Figure 9 presents the CPU consumption of our prototype with two different APKs using the Random Forest classifier. The Grammar Handbook (Figure 9b), which is a malware that simulates a grammar book with a high rate of network packet generation, and the native application of Google Play Store (Figure 9a), representing the goodware class. We select these applications for time analysis for two main reasons: they provide peaks of network traffic near Internet bandwidth of our testbed, at the same time they comprise opposite cases of packet classification decision. Google Play packets were always correctly classified as benign, while fake Grammar Handbook have it most packets classified as malicious, achieving a malware score above the threshold, and thus, a final decision for the APK as malicious. We note that prototype CPU consumption is directly affected by the generated network traffic. This effect, however, was negligible even during the network traffic peaks for both applications. In the worst case, while running our monitoring tool during Google Play downloading an application, CPU consumption remained less than 10% at its peak, with an average of 5%. The number of packets transmitted by both applications is detailed in Table II. Thence, our proposed lightweight network-based malware detection system is feasible for execution in Android devices, with low processing overhead. Moreover, as our technique only demands a small-time window to define the APK class, which in our testbed is defined as 500 seconds interval for APK monitoring, the processing of our model can be even reduced by optimizing this time constant on further research. Therefore, even for network traffic peaks in the moment an application starts, our system were able to maintain low CPU consumption due to the lightweight nature of our detection system.

[7]SimplePerf: https://developer.android.com/ndk/guides/simpleperf



(a) Google Play (Goodware). (b) Grammar Handbook (Malware).

Figure 9: Prototype CPU consumption running on top of a Samsung S9+ and captured packets for the Google Play APK (Goodware) and the Grammar Handbook malware APK. Network events were triggered with user interactivity in both cases. Each time period represents a 10-second interval.

V. LIMITATIONS

Despite achieving substantial results in detecting malicious applications using only network-layer features, we pinpoint limitations of our work, which can serve as motivation for further research on detecting mobile malware regarding network analysis. First, as the dataset generated by our team was based on real APK monitoring, we have faced issues due to amount of time needed to generated data for all APKs, and as consequence, testbed incidents, such as loss of Internet connectivity or drops in Wi-Fi signal during the experiments forced us to repeat the whole experiment many times to guarantee validation of dataset. One way to circumvent this is increasing the parallelism of data generation by augmenting the number of devices collecting data. This would allow our testbed to generate twice or more data than before in the same time period and allow to increase the amount of APKs collected. Second, we defined our time period of classification as the first 500 seconds of the APK execution. Some malware, however, can circumvent detection by only triggering malicious behavior after the 500 seconds mark. In our work, we filtered APKs that did not generated enough data based on a threshold of number of packets and we might have discarded some malware data with this approach. Therefore, optimizations on the capture time period could be performed to increase malware APK traffic. Moreover, a sophisticated malware can be programmed to only badly behaves in a certain period of the day, and thus, further experiments capturing data in different periods of the day should be considered. Other limitation is because when dealing with real data, data labeling is a challenging task. As a preliminary premise of this work, traffic generated by a malicious APK was entirely considered as malicious traffic. However, we observed that this is not always true. We deeply inspected the network traffic generated by malicious APK and we observed portions of benign traffic (from google APIs and third-party servers owned by the application) that are mixed with the untrusted

Table III: Comparison of our proposal with state-of-art research.

Related Work	Analysis Method	Network Features	Machine Learning	Android Impl.	Real Traffic	Lightweight
Andromaly [10]	Static + Dynamic	×	✓	×	×	×
Stream [19]	Dynamic	×	✓	×	✓	×
ProfileDroid [20]	Static + Dynamic	✓(with APK features)	×	✓	✓	×
Yu *et al.* [21]	Dynamic	×	✓	✓	✓	×
Shafiq *et al.* [22]	Dynamic	×	×	×	✓	×
Comar *et al.* [23]	Dynamic	✓	✓	×	✓(ISP traces)	×
DroidSec [24]	Static + Dynamic	×	✓	✓	✓	×
Aresu *et al.* [25]	Dynamic	✓(with appl. layer features)	✓	✓(emulated)	×	×
Arora *et al.* [26]	Dynamic	✓	✓	✓(emulated)	✓	×
Chen *et al.* [27]	Dynamic	✓	✓	✓	×	×
Wang *et al.* [28]	Dynamic	✓(with APK features)	✓(detection on cloud)	✓	✓	✓
This proposal	Dynamic	✓	✓	✓	✓	✓

traffic that is transmitted by the malware APK. This might introduce wrongly labeled data into the classifier training and also impacts the classification accuracy when dealing with benign APKs. A possible solution to circumvent this is, instead of labeling all malware APK traffic as malicious, filter only when destination or origin is different from well-known servers, such as Google servers, or the proprietary server of the application. Despite, we observed most of traffic generated by the malicious APKs used in this research were originated or destined to untrusted sources, i.e., to IP addresses that did not represent well-known domains. Therefore, this limitation did not have significant impact on the research findings and also does not affect the lightweight nature of proposed schemes.

VI. Related Work

Detecting malicious applications in energy-restrained devices is a challenging task. Andromaly is one of the first approaches to perform host-based anomaly detection in energy-constrained devices [10]. The authors combined dynamic and static analysis to obtain 88 features from the operating system, call, messaging services, screen, hardware, among others. Although this work uses an extensive feature set, results of Andromaly are restricted since the authors affirm that at the date of the article there was not enough malware for mobile devices. On the other hand, STREAM employs machine learning algorithms to dynamically classify malware [19]. The authors extracted features from CPU, memory, and binder processes, however, network access was disabled during the malware execution. Similarly, Wei *et al.* proposed *ProfileDroid* [20], a multi-layer detection system for Android Apps. The authors combine four layers of features, including app specification, user interaction, operating system, and network. Nevertheless, the authors disregard malicious application detection. Yu *et al.* employed a machine-learning dynamic analysis of system calls to detect Android malware [21], while Shafiq *et al.* construct a graph from application systems calls and use graph-level feature extraction to detect malware [22]. The authors, however, do not focus on lightweight detection and do not evaluate device performance.

Comar *et al.* proposes a machine-learning-based framework to detect malware with features from layers 3 and 4. The authors use a network operator dataset, which does not represent mobile application traces [23]. Yuan *et al.* proposes an ML-based method with 200 features extracted from both static and dynamic analysis of Android apps for malware detection [24]. The authors achieved high accuracy using a deep-learning model, but they do not provide a lightweight solution, as features are computationally expensive to obtain. Aresu *et al.* analyzed HTTP traffic to detect mobile botnets [25]. According to the authors, 70% of mobile traffic uses the HTTP protocol. The authors created malware clusters using features from GET/POST requests, URL length, among others. The approaches, however, lacks implementation of the detection technique in real Android devices. In the aforementioned works, dataset creation relies on a sandbox or Android emulator to scale data generation. Sophisticated malware, however, is aware of such emulated environment and can inhibit its malicious behavior. Arora and Peddoju proposed an algorithm to reduce the number of features while maximizing the accuracy [26]. The algorithm was validated in Samsung smartphones. Chen *et al.* addressed the problem of malware traffic imbalance by using oversampling techniques with several machine learning algorithms [27]. The authors created a testbed to collect mirrored network traffic on a gateway from a smartphone and a traffic generator machine. Wang *et al.* proposed a lightweight malware detection applying machine learning to network analysis [28]. The authors combine network-layer with application-layer features in a tree-based model. Our system, instead, achieves similar performance using only network-layer features. The authors also claims lightweight because network traffic of the access point is mirrored to a separated cloud infrastructure, where the data processing occurs. Hence, no practical device-level lightweight technique is employed. Furthermore, their scenario restricts the detection system to a specific location where the solution is deployed. The authors also do not evaluate performance regarding different malware families.

We summarize the comparison of our proposal with state-of-art in Table III. To the best of our knowledge, our work is the first to address malware classification with lightweight techniques for network traffic analysis in real Android devices. Moreover, we evaluate our technique using a realistic malware dataset generated on top of Android devices, implement and compare the performance trade-offs of our proposed system and provided a fundamental analysis on network-based malware classification regarding different malware families and variants.

VII. CONCLUSION

Malware detection is a resource-intensive task. While static analysis has drawbacks on efficacy regarding code obfuscation, the literature of dynamic analysis has not taken into account the restrained-resource nature of mobile devices. Therefore, this paper proposed a lightweight network-based Android malware detection system using dynamic analysis. To become lightweight, a feature extractor using a static ring buffer collects only numerical features from the TCP/IP packet header. Then, we apply lightweight machine-learning models, previously trained, to the ring buffer output. Moreover, we implement a prototype of our system in a Samsung Galaxy S9+ with two machine-learning algorithms, Random Forest and AdaBoost. Then, we construct a realistic and up-to-date dataset, comprising network traffic from 359 applications. Results show that both algorithms achieve a good trade-off between classification and performance. Furthermore, we analyzed the capability of our model detecting malware families and malware variants previously unseen by the classifier. With only 14 TCP/IP packet features, our prototype was able to correctly classify more than 90% of APKs with a false positive rate under 3%, without deep packet inspection. Finally, by analyzing two applications that produces lots of packets, our prototype consumed less than 10% of CPU resources even in peaks of 90 Mb. As future work, we will optimize the minimal traffic volume needed to correctly classify an application and scale processing evaluation to a high number of applications.

ACKNOWLEDGMENT

The results presented in this paper were sponsored by Samsung Eletrônica da Amazônia Ltda. under the terms of Brazilian Federal law No. 8.248/91. The authors would like to thank Gabriel Oliveira, Raphael Monteiro, and Anderson Morais for their contributions.

REFERENCES

[1] F. Jejdling, "Ericsson mobility report," Ericsson, Tech. Rep., November 2018, Last Access 05/2019. [Online]. Available: https://www.ericsson.com/assets/local/mobility-report/documents/2018/ericsson-mobility-report-november-2018.pdf

[2] C. Beek et al., "Mcafee labs threats report," McAfee, Tech. Rep., June 2018, Last Access 05/2019. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-jun-2018.pdf

[3] M. Hypponen and T. Tuominen, "F-secure 2017 state of cyber security," F-Secure, Tech. Rep. FS-2017, November 2017, Last Access 05/2019. [Online]. Available: https://blog.f-secure.com/the-state-of-cyber-security-2017/

[4] StatCounter, "Mobile operating system market share world wide," Statcounter GlobalStats, Tech. Rep., November 2018, Last Access 05/2019. [Online]. Available: http://gs.statcounter.com/os-market-share/mobile/world-wide/

[5] K.-W. Lim, S. Secci, L. Tabourier, and B. Tebbani, "Characterizing and predicting mobile application usage," *Computer Communications*, vol. 95, pp. 82 – 94, 2016, mobile Traffic Analytics. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366416301815

[6] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, "The evolution of android malware and android analysis techniques," *ACM Comput. Surv.*, vol. 49, no. 4, pp. 76:1–76:41, Jan. 2017. [Online]. Available: http://doi.acm.org/10.1145/3017427

[7] R. Zachariah, K. Akash, M. S. Yousef, and A. M. Chacko, "Android malware detection a survey," in *IEEE International Conference on Circuits and Systems (ICCS)*. IEEE, Dec 2017, pp. 238–244.

[8] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "Madam: Effective and efficient behavior-based android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, Jan 2018.

[9] N. Peiravian and X. Zhu, "Machine learning for android malware detection using permission and api calls," in *IEEE 25th International Conference on Tools with Artificial Intelligence*, Nov 2013, pp. 300–305.

[10] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Andromaly: A behavioral malware detection framework for android devices," *Journal of Intelligent Inf. Sys.*, vol. 38, no. 1, pp. 161–190, Feb. 2012.

[11] A. G. P. Lobato, M. A. Lopez, I. J. Sanz, A. A. Cardenas, O. C. M. B. Duarte, and G. Pujolle, "An adaptive real-time architecture for zero-day threat detection," in *IEEE International Conference on Communications (ICC)*, May 2018, pp. 1–6.

[12] E. Viegas, A. O. Santin, A. França, R. Jasinski, V. A. Pedroni, and L. S. Oliveira, "Towards an energy-efficient anomaly-based intrusion detection engine for embedded systems," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 163–177, Jan 2017.

[13] S. H. Shaikot and M. S. Kim, "Efficient memory layout for packet classification system on multi-core architecture," in *IEEE Global Communications Conference (GLOBECOM)*, Dec 2012, pp. 2553–2559.

[14] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current android malware," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'17)*. Bonn, Germany: Springer, 2017, pp. 252–276.

[15] Z. Chen, H. Han, Q. Yan, B. Yang, L. Peng, L. Zhang, and J. Li, "A first look at android malware traffic in first few minutes," in *IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 206–213.

[16] P. A. A. Resende and A. C. Drummond, "A survey of random forest based methods for intrusion detection systems," *ACM Comput. Surv.*, vol. 51, no. 3, pp. 48:1–48:36, May 2018.

[17] D. Buhov, M. Huber, G. Merzdovnik, and E. Weippl, "Pin it! improving android network security at runtime," in *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, May 2016, pp. 297–305.

[18] M. Apel, C. Bockermann, and M. Meier, "Measuring similarity of malware behavior," 10 2009, pp. 891–898.

[19] B. Amos, H. Turner, and J. White, "Applying machine learning classifiers to dynamic android malware detection at scale," in *IX International Wireless Communications and Mobile Computing Conference (IWCMC)*, July 2013, pp. 1666–1671.

[20] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Profiledroid: Multi-layer profiling of android applications," in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, ser. Mobicom '12. New York, NY, USA: ACM, 2012, pp. 137–148.

[21] W. Yu, H. Z. L. Ge, and R. Hardy, "On behavior-based detection of malware on android platform," in *IEEE Global Communications Conference (GLOBECOM)*, Dec 2013, pp. 814–819.

[22] Z. Shafiq and A. Liu, "A graph theoretic approach to fast and accurate malware detection," in *IFIP Networking Conference (IFIP Networking) and Workshops*, June 2017, pp. 1–9.

[23] P. M. Comar, L. Liu, S. Saha, P. Tan, and A. Nucci, "Combining supervised and unsupervised learning for zero-day malware detection," in *Proceedings IEEE INFOCOM*, April 2013, pp. 2022–2030.

[24] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: Deep learning in android malware detection," in *Proceedings of the 2014 ACM Conference on SIGCOMM*. New York, NY, USA: ACM, 2014, pp. 371–372.

[25] M. Aresu, D. Ariu, M. Ahmadi, D. Maiorca, and G. Giacinto, "Clustering android malware families by http traffic," in *X International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 2015, pp. 128–135.

[26] A. Arora and S. K. Peddoju, "Minimizing network traffic features for android mobile malware detection," in *Proceedings of the 18th International Conference on Distributed Computing and Networking*, ser. ICDCN '17. New York, NY, USA: ACM, 2017, pp. 32:1–32:10. [Online]. Available: http://doi.acm.org/10.1145/3007748.3007763

[27] Z. Chen, Q. Yan, H. Han, S. Wang, L. Peng, L. Wang, and B. Yang, "Machine learning based mobile malware detection using highly imbalanced network traffic," *Inf. Sciences*, vol. 433, pp. 346–364, 2018.

[28] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, and Z. Jia, "A mobile malware detection method using behavior features in network traffic," *Jrnl. of Network and Comp. Applications.*, vol. 133, pp. 15 – 25, 2019.