A Friend or a Foe?
Detecting Malware using Memory and CPU Features

Jelena Milosevic, Miroslaw Malek and Alberto Ferrante
ALaRI, Faculty of Informatics, Universitá della Svizzera Italiana, Lugano, Switzerland

Keywords: Malware Detection, Dynamic Detection, Android, Internet of Things (IoT).

Abstract: With an ever-increasing and ever more aggressive proliferation of malware, its detection is of utmost importance. However, due to the fact that IoT devices are resource-constrained, it is difficult to provide effective solutions.

The main goal of this paper is the development of lightweight techniques for dynamic malware detection. For this purpose, we identify an optimized set of features to be monitored at runtime on mobile devices as well as detection algorithms that are suitable for battery-operated environments. We propose to use a minimal set of most indicative memory and CPU features reflecting malicious behavior.

The performance analysis and validation of features usefulness in detecting malware have been carried out by considering the Android operating system. The results show that memory and CPU related features contain enough information to discriminate between execution traces belonging to malicious and benign applications with significant detection precision and recall. Since the proposed approach requires only a limited number of features and algorithms of low complexity, we believe that it can be used for effective malware detection, not only on mobile devices, but also on other smart elements of IoT.

1 INTRODUCTION

Malicious software, shortly *malware*, is any type of program that is created to cause harm, ranging from minor inconvenience to loss of data. Among others, it has the goal of stealing sensitive information from users, taking control over the operating system, and damaging or even completely disabling the device.

The focus of Internet security is shifting from the desktop and the data center to the home, the pocket, the purse, and, ultimately, the infrastructure of the Internet itself Symantec Corporation (2015). Furthermore, for the second consecutive year, mobile devices are perceived as IT security's weakest link Group (2015). In fact, with increased number of mobile devices and their widespread usage, mobile malware is spreading as well. Threat alerts for Android constantly grew year by year and, according to McAfee Labs McAfee Labs (2015), the collection of mobile malware continued its steady climb as it broke 6 million samples in the fourth quarter of 2014, up 14% over the third quarter of the same year.

Google Android is certainly one of the main operating systems for smart devices, with its 85% market share worldwide for smartphones Gartner, Inc. (2015). Android was first developed for mobile devices, but it has evolved into a pervasive operating system, being also available for wearable devices, TV sets, and vehicles. Google has also announced *Brillo*, an embedded operating system based on Android and aimed at being used on low-power and memory constrained IoT devices Google Developers (2015). In this paper, due to the availability of large number of samples, we have chosen Android malware for our case study, even though we aim at developing detection algorithms suitable also for resource-constrained IoT devices. The aforementioned operating system convergence towards resource-constrained devices will facilitate the direct porting of these results from one system to the other.

Mobile malware detection is used in the battery-operated environment of mobile devices, where energy is the main limiting factor to run computationally complex algorithms, as opposed to regular desktop malware detection systems. However, same as for PCs, detection performance has to be high in order to minimise the number of false positives that may cause disturbance to users. Increased number of malicious samples and malware families in recent years, make the efficient detection a challenging problem and its

Milosevic, J., Malek, M. and Ferrante, A.
A Friend or a Foe? Detecting Malware using Memory and CPU Features.
DOI: 10.5220/0005964200730084
In *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016) - Volume 4: SECRYPT*, pages 73-84
ISBN: 978-989-758-196-0

solutions are mostly trade-offs between the mentioned requirements. Most of the existing solutions are either with good detection performance, but too complex to be used at run-time and within limited energy budget, or lightweight but with limited ability to detect the plethora of malware ever newly generated.

Currently, there are two main approaches to detect malware: static and dynamic. Static detection is based on the investigation of the applications static features (e.g., permissions, manifest files, API calls). It is usually lightweight and suitable for the limited resources of mobile devices. The problems with static detection are that it cannot cope with increased number of malicious samples and their variations, it is prone to obfuscation and alone may no longer be sufficient to identify malware Moser et al. (2007). What appears as a promising candidate is dynamic malware detection that is based on observing systems at run-time. In such a way, malware cannot easily obfuscate. However, such detection can be too complex for battery-operated mobile environments and lightweight solutions are still needed.

A promising approach towards efficient dynamic mobile malware detection is the identification of a limited set of features that provide the ability, through suitable algorithms, to discriminate between benign and malicious behavior. The features in this set are the ones that are monitored at runtime, and are indicative for detecting presence of malware. Limiting the number of used features diminishes the requirements for resources (memory, CPU, and energy), thus lowering computational overhead. In this paper, we use term feature to describe an individual measurable property of a phenomena being observed Bishop (2006). In the literature, also different terms, such as attributes, variables, and parameters, can be found to describe the same property.

We focus on the usefulness of memory and CPU related features for efficient malware detection. Namely, our main hypothesis, confirmed by the results that we have obtained, is that by observing only memory and CPU related features, malicious execution traces can be distinguished from benign ones by means of algorithms of low complexity. We identify an optimized set of memory and CPU related features and prove their effectiveness in detecting malware by applying different lightweight classification algorithms: Naive Bayes, Logistic Regression, and J48 Decision Tree. The experiments we performed in this work are with a set of malicious and benign applications running on Android mobile devices. However, due to the simplicity of monitoring and the low-complexity of algorithms used to detect malware, we are confident that our approach can also be used in majority of other IoT devices.

The rest of the paper is organized as follows. First, in Section 2 the related work is discussed. Then, in Section 3 the proposed approach is outlined. Later, in Section 4 the performed experiments are described, followed by detailed explanation of the obtained results in Section 5 and discussion in Section 6.

2 RELATED WORK

With increased number of mobile malware threats the need to protect mobile devices from them is growing, resulting in higher demands for effective detection systems and increased research activity in this area. Most of the existing malware is developed for mobile devices. Not surprisingly, most of the available solutions are focused on these devices as well. We report related approaches where observation of different features was used for mobile malware detection, both static and dynamic, and compare our approach with them.

An effective static approach to malware detection is proposed in Arp et al. (2014) where high detection quality is achieved by using features from the manifest file and feature sets from disassembled code; detection is performed by means of Support Vector Machines. Also, the mechanism presented in Wu et al. (2012) uses static features including permissions, Intent messages passing and API calls to detect malicious Android applications, with K-means being the best performing clustering algorithm. However, the main disadvantage of these approaches, since they are based on static analysis, is the inability to detect malware at run-time and, thus, the inability to detect malicious behavior after the installation of applications and/or in between periodic system scans.

Another static approach is presented in Enck et al. (2009) where the authors propose to identify malware based on sets of permissions. In Felt et al. (2011a), sending SMS messages without confirmation or accessing unique phone identifiers like the IMEI are identified as promising features for malware detection. Legitimate applications ask those permissions less often Felt et al. (2011b). Still, using only asked permissions, as it is done in Felt et al. (2011a,b) produces a high false positive rate.

In Truong et al. (2013), as a static feature for detecting susceptibility of a device to malware infection, a set of identifiers representing the applications on a device is used. The assumption is that the set of applications used on a device may predict the likelihood of the device being classified as infected in the future. Nevertheless, observing just this feature is not enough to give satisfactory answer about device being attacked due to relatively low precision and recall Truong et al. (2013).

Battery power, as a dynamic feature, is also considered for detection of malware. One of the proposed solutions, VirusMeter Liu et al. (2009) monitors and audits power consumption on mobile devices with a power model that accurately characterizes power consumption of normal user behaviors. Also in Kim et al. (2008) the authors use power monitoring to detect malware. They first extract characteristic power consumption signatures of malicious and benign applications and then, while using the system, detect if its power consumption is more similar with benign or malicious sample. However, althought it appears as a promising feature, to what extent malware can be detected on phones by monitoring just battery power remains an open research question Becher et al. (2011).

In Shabtai et al. (2012), Andromaly, a framework for detecting malware, is proposed. It uses a variety of dynamic features related to: touch screen, keyboard, scheduler, CPU load, messaging, power, memory, calls, operating system, network, hardware, binder, and leds. The authors of Andromaly compared False Positive Rate, True Positive Rate, and accuracy of different detection algorithms and concluded that the algorithms that outperformed the others were Logistic Regression and Naive Bayes. The results were obtained using 40 benign applications and four malicious samples, developed by the authors, since real malicious samples did not exist at that time.

In Ham and Choi (2013), feature selection was performed on a set of run-time features related to network, SMS, CPU, power, process information, memory, and virtual memory. Four different classification algorithms were evaluated and as a measure of features usefulness Information Gain was used. Random Forest was the one that provided the best performance. Random Forest is a combination of different tree classifiers Breiman (2001). Although it is a powerful algorithm in achieving high quality of detection, it also has high complexity, making it unsuitable for mobile devices. Additionally, results have been obtained by only considering 30 benign and five malicious applications, with a limited coverage of the high variety of malware available to date. This may limit the applicability of results, both in terms of algorithm and of selected features.

As previously mentioned, Shabtai et al. (2012) and Ham and Choi (2013) also consider some memory and CPU features as part of a larger feature set. While we have limited our observation to memory and CPU related features, we have considered more detailed features related to single applications, instead of generic ones such as total CPU or memory usage. In our opinion, our approach better covers CPU and memory behavior of the observed applications. Additionally, as opposed to Shabtai et al. (2012), our approach uses real malicious samples with higher variety of malicious behavior, and less complex detection methods rather than Random Forest, as it is done in Ham and Choi (2013).

Another recent work, presented in Milosevic et al. (2016), also takes into account memory and CPU features and their usefulness in malware detection. It analyzes these features and their significance within the malware families they belong to, and takes into account the most indicative ones for each family. It concludes that some features appear as good candidates for malware detection in general, some features appear as good candidates for detection of specific malware families, and some others are simply irrelevant. For the analysis of usefulness of features, the authors use Principal Component Analysis. However, in this work further validation of features by using detection algorithms is not performed nor benign samples are included in the experiments.

As opposed to the approach proposed in Milosevic et al. (2016), we validate the usefulness of the selected features by using three different detection algorithms and use a balanced dataset consisting of both benign and malicious applications. Additionally, in order to find the most indicative features, we take into account five different feature selection methods. Finally, we focus on discrimination of malicious and benign execution traces, while the paper deals with usefulness of features within different malware families.

3 METHODOLOGY

Towards enabling efficient dynamic detection of mobile malware, we propose the following approach to identify the most indicative features related to memory and CPU to be monitored on mobile devices and the most appropriate classification algorithms to be used afterwards:

1. Collection of malicious samples, representing different families, and of benign samples
2. Execution of samples and collection of the execution traces
3. Extraction of features, from the execution traces, related to memory and CPU
4. Selection of the most indicative features

5. Selection of the most appropriate classification algorithms
6. Quantitative evaluation of the selected features and algorithms

The first step defines the dataset to be used in the remaining parts of the methodology. Namely, it is important to use malicious samples coming from different malware families, so that the diverse behavior of malware is covered to as large extent as possible. Furthermore, it is needed to set up the execution environment to run malicious samples, so that malicious behavior can be triggered. Additionally, such environment should provide the possibility to execute large number of malicious samples, within reasonable time, so that the obtained results have statistical significance. We have achieved these requirements, first, by using a variety of malware families with broad scope of behavior, second by triggering different events while executing malicious applications and, third, by using an emulation environment that enabled us to execute applications quickly. Since our goal is to discriminate between malicious and benign execution records, we have taken into account also benign samples, and executed them in same conditions used for malicious applications. While usage of an emulator enables us to execute statistically significant number of applications on one hand, on the other hand it is our belief that its usage instead of a real device has a limited or no effect on results, due to the nature of features observed. However, we are aware of the fact that the use of an emulator may prevent the activation of certain sophisticated malicious samples.

Before identification of the most indicative symptoms, feature extraction and selection needs to be performed. A comprehensive survey covering the state-of-the-art in feature selection can be found in Liu and Yu (2005). Intuitively, a better classification result can be achieved if we add more features to the dataset, however this is not the case, as it can happen that irrelevant and redundant features confuse classifiers and decrease detection performance. Due to this fact, we have performed feature selection as a separate step, in which we have evaluated features usefulness based on the following techniques: Correlation Attribute Evaluator, CFS Subset Evaluator, Gain Ratio Attribute Evaluator, Information Gain Attribute Evaluator, and OneR Feature Evaluator Hall et al. (2009). We have chosen feature selection techniques due to their difference in observing usefulness of features as they are based either on statistical importance or information gain measure. Following, the list of feature selection algorithms that we have used:

- *Correlation Attribute Evaluator* calculates the worth of an attribute by measuring the correlation between it and the class.
- *CfsSubsetEval* calculates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them. Subsets of features that are highly correlated with the class while having low intercorrelation are preferred Hall (1998).
- *Gain Ratio Attribute Evaluator* calculates the worth of an attribute by measuring the gain ratio with respect to the class.
- *Information Gain Attribute Evaluator* calculates the worth of an attribute by measuring the information gain with respect to the class.
- *OneR Feature Evaluation* calculates the worth of an attribute by using the OneR classifier, which uses the minimum-error attribute for prediction Holte (1993).

In order to validate the usefulness of selected features we have used the following detection algorithms having different approach to detection: Naive Bayes, Logistic Regression, and J48 Decision Tree. While most of the steps of the proposed approach are executed offline (i.e., on machines equipped with extensive computational resources), the classification algorithm will be executed on the mobile devices; thus, it needs to be compatible with their limited resources. This is the reason why we also take into account the complexity of algorithms, and use the ones with low complexity. A brief description of aforementioned classification algorithms follows:

- *Naive Bayes* is a probabilistic classifier. It applies Bayes theorem making an assumption that the features are independent. Under this *naive* assumption it calculates probability of an unknown instance belonging to each class and selects the one with the highest probability as an output John and Langley (1995).
- *Logistic Regression* is a linear classifier that calculates the conditional probabilities of possible outcomes and chooses the one with the maximum likelihood. It is a technique widely used in many fields.
- *A decision tree-based J48 classifier* is an open source implementation of C4.5 algorithm Hall et al. (2009); C4.5 is a statistical classifier that for each node of the tree, chooses the attribute of the data that the most effectively splits its samples into subsets; the splitting criterion being maximum information gain Quinlan (1993).

4 EVALUATION

We have executed both malicious and benign samples by using the Android SDK emulator and we have observed their influence on 53 features representing phone state. We have analysed collected execution traces by applying methods to reduce the set of significant features used in the classifiers, based both on statistical significance of features and on information gain measure. The selected features, extracted from the training set, were afterwards employed to train different classifiers for a comparative evaluation; these classifiers are then used to categorize execution records into malicious or benign. To build a balanced set of applications, we used malicious samples coming from Malware Genome Project Zhou and Jiang (2012) and benign samples coming from Play Store Google Inc. (2015b).

In this section, we describe in detail the dataset that we have used, and the experiments that we have performed. Furthermore, we list the features that we have collected, describe the settings of the techniques used for feature analysis, so as implementation details of the classification algorithms used for malware detection.

4.1 Dataset

The dataset consists of a mix of malicious and benign applications. The Malware dataset is composed of 1,247 malicious applications taken from the Malware Genome project Zhou and Jiang (2012) and belonging to 41 families. These malware families cover a broad range of malicious behaviors, including different installation methods (repackaging, update, drive-by download, standalone), activation mechanisms (Boot Completed, Phone Events, SMS, Calls, USB, Battery, Package, Network, System Events) and different malicious payloads (privilege escalation, remote control, financial charges, and personal information stealing). More details on each of these families are provided in Zhou and Jiang (2012).

The benign dataset has been obtained by running 952 benign applications downloaded from the Google Play Store Google Inc. (2015b). We believe that the applications coming from Google Play Store can be used as benign due to the detailed analysis done to identify potentially harmful applications and the low infection rate reported in Google Inc. (2015c). We are aware that some of these applications could potentially contain malicious behavior, but we believe that the number of benign ones is still much higher making the obtained dataset representative.

4.2 Execution Environment

Memory and CPU usage traces were recorded by running the applications, one at a time, on the Android emulator; each application has been run for 10 minutes, even though some of the traces are shorter, due to emulator hiccups. The considered monitoring interval is equal to two seconds. Although it could be the case that a longer execution period would provide us more significant results, we believe that the duration that we have chosen is a good trade-off between time when most of the malware samples expose their malicious intents and duration of the overall experimentation.

The Android emulator of choice was the one included in the Android Software Development Kit Android Open Source project (2015b) release 20140702, running Android 4.0. The reason why an Android emulator was chosen instead of real devices is that this solution minimizes the time necessary to set up each application before running, thus providing the ability to run a large number of applications, making the obtained dataset more significant. The Android operating system was each time re-initialized before running each application, so that the interferences (e.g., changed settings, running processes, and modifications of the operating system files) from previously run samples are avoided.

All this process was automated by means of a shell script that was run on a Linux PC and made use of Android Debug Bridge (adb) Android Open Source project (2015a), a command line tool that lets the PC communicate with an emulator instance or with an Android device. The Monkey application exerciser Android Open Source project (2015c) was also used in the script. The Monkey is a command-line tool that can be run on any emulator instance or on a device; it sends a pseudo-random stream of user events into the system, which acts as a stress test on the application software. In our script, Monkey was used to activate different features of the applications; adb was used to monitor application features, namely the memory usage of the considered sample, as well as to install the applications. In summary, for each application, the following actions were performed:

- A clean install of the Android operating system
- The application installation on the Android Emulator by means of adb
- The memory monitoring by means of periodic calls to the *adb shell* command
- Initialization and run of the application for 10 minutes by using the Monkey.

4.3 Collected Features

We have taken into account all the features related to memory and CPU that can be accessed in Android. In total, this makes 53 features for each running application; these features are listed in Table 1. There are five features related to CPU: three related to CPU usage, and two to virtual memory exceptions (major and minor faults). The remaining 48 features are related to different aspects of memory usage.

4.4 Feature Analysis Techniques

As described in Section 3, collected features are analysed by using Correlation Attribute Evaluator, CFS Subset Evaluator, Gain Ratio Attribute Evaluator, Information Gain Attribute Evaluator, and OneR Feature Evaluator. For the mentioned feature selection methods, their implementation in Weka data mining tool Hall et al. (2009) has been used. More in detail, CFS subset evaluation is performed by using Greedy forward search through the space of attribute subsets Hall (1998). Correlation Attribute Evaluator, Gain Ratio Attribute Evaluator, Information Gain Attribute Evaluator, and OneR Attribute Evaluator are used together with Ranker method that ranks attributes by their individual performance.

4.5 Detection Algorithms

As discussed in Section 3, the classifiers of choice are: Naive Bayes, Logistic Regression, and J48 Decision Tree. Weka implementations of these algorithms have been used. Naive Bayes has been set assuming normality and modelling each conditional distribution with a single Gaussian; Logistic Regression has been used with ridge estimator, since it has been shown that it can improve attribute estimation and decrease the error made in further predictions Le Cessie and Van Houwelingen (1992). J48 has been used with pruning, setting the confidence factor used for pruning to 0.25 and the minimum number of instances per leaf to 5000; these values represent a trade-off between number of instances in the dataset, speed of execution and the quality of classification.

4.6 Evaluation of the Approach

The validation of the classification is done by using ten-fold cross validation, that is widely used model validation technique that can estimate how accurately observed model will perform in practise Kohavi (1995). In case of ten-fold cross validation, the dataset is divided into ten parts, where at each round one part, consisting of nine folds, is considered as a training set and the remaining part is used as a test set. This procedure is repeated ten times, each time using a different training and a test set, and after ten rounds the average detection performance is reported.

Table 1: List of all the considered features; totals are related only to single applications.

CPU Usage	Total CPU Usage User CPU Usage Kernel CPU Usage
Virtual Memory	Page Minor Faults Page Major Faults
Native memory	Native PSS Native Shared Dirty Native Private Dirty Native Heap Size Native Heap Alloc Native Heap Free
Dalvik memory	Dalvik PSS Dalvik Shared Dirty Dalvik Private Dirty Dalvik Heap Size Dalvik Heap Alloc Dalvik Heap Free Dalvik Cursor PSS
Cursor memory	Cursor Shared Dirty Cursor Private Dirty
Android shared memory	Ashmem PSS Ashmem Shared Dirty Ashmem Private Dirty
Memory-mapped native code	.so mmap PSS .so mmap Shared Dirty .so mmap Private Dirty
Memory mapped Dalvik code	.dex mmap PSS .dex mmap Shared Dirty .dex mmap Private Dirty
Memory-mapped fonts	.ttf mmap PSS .ttf mmap Shared Dirty .ttf mmap Private Dirty
Other memory-mapped files and devices	.jar mmap PSS .jar mmap Shared Dirty .jar mmap Private Dirty .apk mmap PSS .apk mmap Shared Dirty .apk mmap Private Dirty Other mmap PSS Other mmap Shared Dirty Other mmap Private Dirty
Non-classified memory allocations	Unknown PSS Unknown Shared Dirty Unknown Private Dirty Other dev PSS Other dev Shared Dirty Other dev Private Dirty
Memory totals	Total PSS Total Shared Dirty Total Private Dirty Total Heap Size Total Heap Alloc Total Heap Free

5 RESULTS

Results show that by observing limited number of features related to memory and CPU (only seven in case of Naive Bayes and Logistic Regression, and six in case of J48 Decision Tree), the execution traces belonging to malicious applications can be identified with precision and recall of more than 84%. In our opinion, this result is valuable, having in mind the small number of features used, the lightweight classification algorithms, and the good F-measure score obtained.

To rank the available features, we applied the five feature selection methods introduced in Section 3; the first fifteen highest ranked features for each method are shown in Table 2. We may notice that different ranking methods provide different results; however, a number of features related to memory mapping, such as *.ttf mmap PSS* and *.so mmap PSS*, are among the highest ranked ones in most methods. In Table 2 for CFS Subset Evaluator only seven features are listed, without any ranking. This is due to the fact that CFS does not evaluate single features but, instead, subsets of features, calculating their usefulness with respect to the other subsets.

As discussed in Section 4, we have used three different classification algorithms. In case of Naive Bayes and Logistic Regression we have evaluated the results of the classifiers both considering all the available features and by considering only the highest ranked ones for each feature selection method. On the obtained dataset, we have performed ten-fold cross validation and have taken into account precision, recall, and F-measure of a model as a result of the experiment. We have repeated this procedure for all five feature selection methods taken into account.

For J48 Decision Tree, we have trained the model by changing the maximum number of instances that a node can have and by observing its quality of detection (precision, recall, and F-measure). The number of instances is important in order to avoid overfitting, a situation in which the decision tree consists of many nodes that are very well fitted for the training set, but do not perform well on a test set and unseen data. In J48 Decision Tree, the maximum number of instances in the three can be set, but meaningful features are selected by the algorithms and cannot be set externally.

Table 3 summarizes the results obtained with the different classifiers. The initial model includes all the 53 features available. The second model, instead, is the one providing maximum F-measure; the last one provides the best ratio between F-measure and the number of features considered. We use F-measure (sometimes also called F-score) as a quantitative description of the quality of detection models, since it includes both precision (the fraction of detected instances that are relevant) and recall (the fraction of relevant instances that are detected).

In case of Naive Bayes, with the dataset containing all 53 features, an F-measure of 0.77 is obtained. After taking into account rankings of the features for all five feature selection methods, we identified an optimized set of features, both with respect to their number and to F-measure, as the one computed with the CFS Subset Evaluator. As it can be seen from Table 3, the obtained F-measure in this case is 0.83, with a set of seven features.

In case of Logistic Regression, using all features, we obtained F-measure of 0.83. However, decreasing the number of features and observing system performance, we obtained an increased F-measure of 0.86, obtained by using Correlation Attributes with rankings higher than 0.03 (38 features). The optimal model with respect to both number of features and quality of detection is again obtained by using features selected by CFS Subset Evaluator. This model has an F-measure of 0.84.

For what concerns J48, the obtained model, that uses six features, has an F-measure of 0.82. The decision three, with the selected features thereby used, is shown in Figure 1.

In summary, features that are the most indicative for the Logistic Regression and Naive Bayes classifiers are the seven listed in the *CFS Subset Evaluator* column of Table 2; for J48 Decision Tree, the six most significant features are listed in Figure 1. A brief description of these features is provided in Table 4. The best detection performance with respect to F-measure, so as with respect to the ratio between F-measure and the number of features considered, is achieved by the Logistic Regression algorithm. The best F-measure is obtained by considering 38 highest ranked features provided by the Correlation Attribute Evaluator; the optimized set of only 7 features is obtained by considering the CFS Subset Evaluator. Also the other two detection algorithms considered, namely Naive Bayes and J48 Decision Tree, perform well, showing similarly good detection performance.

6 DISCUSSION

We propose and validate an approach for detection of malicious execution traces that takes into account limited resources of battery-operated devices. This is the case due to the small number of features that is used and the lightweight algorithms that are taken into account for detection. The discussed algorithms

Table 2: Top 15 features ranked by different feature selection algorithms.

Correlation Attribute Evaluator		**CFS Subset Evaluator**	**Gain Ratio Attribute Evaluator**	
Ranking	*Feature*	*Feature*	*Ranking*	*Feature*
0.49	.ttf mmap PSS	.so mmap Shared Dirty	0.12	.jar mmap PSS
0.46	.so mmap PSS	.jar mmap PSS	0.06	.dex mmap Private Dirty
0.43	.dex mmap PSS	.ttf mmap PSS	0.05	Other dev Private Dirty
0.39	TOTAL PSS	.dex mmap PSS	0.04	.dex mmap PSS
0.38	.so mmap Shared Dirty	.dex mmap Private Dirty	0.04	.ttf mmap PSS
0.36	TOTAL Private Dirty	Other mmap Private Dirty	0.04	.so mmap Shared Dirty
0.36	.jar mmap PSS	CPU Total	0.03	.so mmap PSS
0.33	Unknown Private Dirty		0.03	.so mmap Private Dirty
0.33	Unknown PSS		0.03	Other mmap Private Dirty
0.31	CPU User		0.03	.dex mmap Shared Dirty
0.30	CPU Total		0.03	Unknown PSS
0.30	TOTAL Heap Size		0.03	Native Heap Size
0.29	TOTAL Heap Alloc		0.03	Unknown Private Dirty
0.27	Other mmap PSS		0.03	Other mmap Shared Dirty
0.26	Native Heap Alloc		0.02	TOTAL PSS

Information Gain Attribute Evaluator		**OneR Attribute Evaluator**	
Ranking	*Feature*	*Ranking*	*Feature*
0.28	.dex mmap PSS	79.99	.ttf mmap PSS
0.27	.ttf mmap PSS	79.64	.dex mmap PSS
0.25	Unknown PSS	78.92	Unknown PSS
0.25	.so mmap PSS	78.51	.so mmap PSS
0.23	Native Heap Size	76.24	TOTAL Heap Size
0.23	TOTAL Heap Size	76.19	Native Heap Size
0.21	Unknown Private Dirty	75.87	Unknown Private Dirty
0.21	.so mmap Private Dirty	75.44	.so mmap Private Dirty
0.17	.so mmap Shared Dirty	73.25	.so mmap Shared Dirty
0.17	Ashmed PSS	71.26	Native PSS
0.15	Other mmap PSS	71.21	Other mmap PSS
0.15	TOTAL PSS	71.17	Ashmed PSS
0.14	minor faults	70.75	minor faults
0.13	Native PSS	69.78	TOTAL PSS
0.13	TOTAL Private Dirty	69.49	.apk mmap PSS

are envisioned to be part of a run-time malware detection system installed on mobile devices. In such system, identified optimized features will be periodically monitored, and the classification of the executions traces will be done in the way proposed in this work. The classified execution traces will be further analysed at the application level and in case of a detected malicious application, the user will be notified. While in this paper we focused on the classification of the executions traces, the classification of the entire applications is envisioned as a next step of our work. We believe that such a lightweight detection system could be used either as a complement to other detection systems (e.g., static ones) or in two-level detection systems, such as the one presented in Milosevic et al. (2014), to raise the first alarm.

6.1 Highlights of the Approach

The approach adopted in developing the detection method presented in this paper, provides the ability to underline a number of key facts as follows.

From the results of our experiments, we can claim that CPU and memory features contain useful information for discriminating between the execution traces related to malicious or benign applications. Additionally, good detection performance can be obtained while using algorithms of low complexity, suitable for battery-operated mobile devices.

We have shown that in case of Logistic Regression and Naive Bayes, detection performance, measured with precision, recall, and F-measure, and presented in Table 3 is improved even if the number of features is decreased from 53 to 7. Reducing the number of features without reducing performance of the system is in line with our intention to design effective, while at the same time efficient, mobile detection system.

Out of five feature selection methods taken into account, our analysis shows that CFS Subset Evalua-

Table 3: Performance of the classifiers when different number of features are considered.

		Model		
		Initial	**Max. F-measure**	**Optimized**
Naive Bayes	Precision	0.79	0.84	0.84
	Recall	0.76	0.83	0.83
	F-measure	0.77	0.83	0.83
	No. of features	53	7	7
Logistic Regression	Precision	0.84	0.86	0.84
	Recall	0.84	0.86	0.84
	F-measure	0.83	0.86	0.84
	No. of features	53	38	7
J48 Decision Tree	Precision	–	–	0.83
	Recall	–		0.83
	F-measure	–	–	0.82
	No. of features	–	–	6

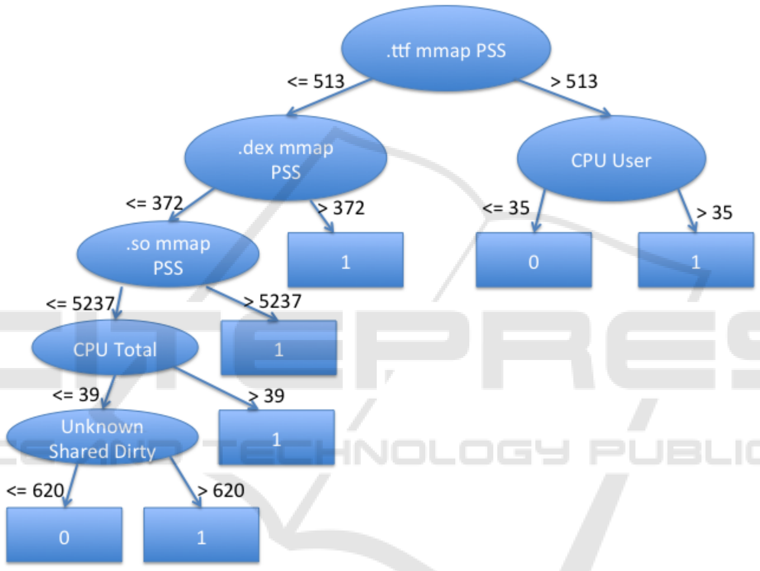

Figure 1: Optimized J48 Decision Tree; malicious and benign traces are labelled with *0* and *1*, respectively.

tor performs the best and identifies the set of features that demonstrates the highest detection performance among all considered detection algorithms.

Out of three detection algorithms taken into account, Logistic Regression has the highest detection performance measured by F-measure. Therefore, we have shown that a very small number of features, coupled with algorithms of linear complexity in the number of features, can be used to detect malware effectively, making the approach lightweight and potentially compatible with the limited resources of small IoT nodes.

We have observed that malicious samples consume less memory and CPU compared to the benign ones. This can be seen looking into values of features of J48 Decision Tree presented in Figure 1. A similar behavior can also be observed by looking into details of the obtained Naive Bayes and Logistic Regression models. The reason for this could be that malicious applications perform only a limited activity, connected to their malicious intent, rather than the legitimate actions for which they were installed by the user.

Ten-fold cross validation was used to evaluate performance of the different classifiers and of the selected sets of features. This approach is equivalent to testing by applying a mix of execution traces (or parts of them) related to both malicious samples that have been used during training and others that have not. This situation is still worse than what happens in reality, where the large majority of malicious samples are known to the classifiers and only a fraction of them is not.

We have performed validation by using execution

Table 4: Brief description of the most indicative features Google Inc. (2015a).

Feature Name	Feature Description
.so mmap Shared Dirty	Shared memory, in the *Dirty* state, being used for mapped native code. The *Dirty* state is due to fix-ups to the native code when it is loaded into its final address.
.so mmap PSS	Memory used for native code, including platform code shared across applications
.jar mmap PSS	Memory usage for Java archives, including pages shared among processes
.ttf mmap PSS	Memory usage for true type fonts, including pages shared among processes
.so mmap PSS	Memory used for Dalvik or ART code, including platform code shared across applications
.so mmap Private Dirty	Private memory, in the *Dirty* state, being used for mapped Dalvik or ART code. The *Dirty* state is due to fix-ups to the native code when it is loaded into its final address.
Other mmap Private Dirty	Private memory used by unclassified contents that is in the *dirty* state
Unknown shared dirty	Shared memory that in the *dirty* state that cannot be classified into one of the other more specific items
CPU User	Userspace CPU usage by the considered application
CPU Total	Total (User + System) CPU usage by the considered application

traces obtained when only one application was running: this situation is very likely to be the most common one in IoT nodes with limited resources.

6.2 Limitations of the Approach

Our detection system uses dynamic features that are by their nature more difficult to evade than the static ones. The assumption for usage of dynamic features (in our case memory and CPU) is that even when the applications are repackaged or obfuscated, during their execution they will still have similar behavioral footprints, and a classifier trained on these features could properly discriminate malicious applications from benign ones. Our detection method can detect effectively known malicious samples and unknown malicious samples belonging to known malware families. However, it cannot guarantee, as any other detection method, absolute security. If a malicious attacker develops a new malicious sample that has a behavior reflected in significantly different feature profiles than the ones observed in our dataset, the detection system might not be able to detect it.

Our approach is data driven and, as other methods that use a similar approach, such as Shabtai et al. (2012)Ham and Choi (2013), it does not provide any help in understanding how the selected features and their relations are connected with the behavior of the applications. Therefore, while selected features can be described individually and the rationale beyond algorithms can be explained, an explanation on why features are important for the identification of malware is possible only in the simplest cases, where the number of available features is very limited and the system is extremely simple. However, without usage of these mathematical algorithms and models it would be impossible to measure usefulness of features, identify their correlations, and choose only the most indicative ones.

7 CONCLUSIONS

The usefulness of CPU and memory features in the discrimination between malicious and benign execution traces has been investigated. We use different feature selection methods based on statistics and information theory and identify the most indicative ones. Furthermore, in order to validate the usefulness of the features we use classification algorithms that are suitable for malware detection in the limited battery-operated environment of mobile and other IoT devices. Results show that memory and CPU features contain enough information to discriminate between benign and malicious execution behavior of applications. In particular, only six or seven features, depending on the classification algorithm, are sufficient to identify execution traces that can be associated with malware with a precision and recall that are higher than 84%. Since the proposed methodology is of low complexity, it is also applicable to a wide range of resource-constrained devices in Internet of Things.

REFERENCES

Android Open Source project (2015a). Android Debug Bridge. Online: http://developer.android.com/tools/help/adb.html.

Android Open Source project (2015b). Android Software Development Kit. Online: https://developer.android.com/sdk/index.html.

Android Open Source project (2015c). UI/Application Exerciser Monkey. Online: http://developer.android.com/tools/help/monkey.html.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., and Rieck, K. (2014). DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *NDSS*.

Becher, M., Freiling, F. C., Hoffmann, J., Holz, T., Uellenbeck, S., and Wolf, C. (2011). Mobile Security Catching Up? Revealing the Nuts and Bolts of the Security of Mobile Devices. In *Symposium on Security* *and Privacy*, SP '11, pages 96–111. IEEE Computer Society.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

Enck, W., Ongtang, M., and McDaniel, P. (2009). On lightweight mobile phone application certification. In *16th ACM conference on Computer and communications security (CCS)*, pages 235–245. ACM.

Felt, A. P., Finifter, M., Chin, E., Hanna, S., and Wagner, D. (2011a). A Survey Of Mobile Malware in the Wild. In *1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM)*, pages 3–14. ACM.

Felt, A. P., Greenwood, K., and Wagner, D. (2011b). The effectiveness of application permissions. In *2nd USENIX conference on Web application development (WebApps)*, pages 7–7. USENIX Association.

Gartner, Inc. (2015). Gartner says emerging markets drove worldwide smartphone sales to 15.5 percent growth in third quarter of 2015. Online: http://www.gartner.com/newsroom/id/3169417.

Google Developers (2015). Brillo. Online: https://developers.google.com/brillo.

Google Inc. (2015a). Android Developers – Investigating Your RAM Usage. Online: http://developer.android.com/tools/debugging/debugging-memory.html.

Google Inc. (2015b). Google Play. Online: https://play.google.com.

Google Inc. (2015c). Google Report – Android Security 2014 Year in Review. Technical report. Online: https://static.googleusercontent.com/media/source.android.com/it//devices/tech/security/reports/Google_Android_Security_2014_Report_Final.pdf.

Group, C. (2015). 2015 cyberthreat defense report. Technical report. Online: http://www.brightcloud.com/pdf/cyberedge-2015-cdr-report.pdf.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, 11(1):10–18.

Hall, M. A. (1998). *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, Hamilton, New Zealand.

Ham, H.-S. and Choi, M.-J. (2013). Analysis of android malware detection performance using machine learning classifiers. In *ICT Convergence (ICTC), 2013 International Conference on*, pages 490–495.

Holte, R. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11(1):63–90.

John, G. and Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In *In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann.

Kim, H., Smith, J., and Shin, K. G. (2008). Detecting energy-greedy anomalies and mobile malware variants. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, MobiSys '08, pages 239–252, New York, NY, USA. ACM.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. pages 1137–1143. Morgan Kaufmann.

Le Cessie, S. and Van Houwelingen, J. C. (1992). Ridge estimators in logistic regression. *Applied statistics*, pages 191–201.

Liu, H. and Yu, L. (2005). Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491–502.

Liu, L., Yan, G., Zhang, X., and Chen, S. (2009). VirusMeter: Preventing Your Cellphone from Spies. In *12th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 244–264. Springer.

McAfee Labs (February 2015). Threats Report. Technical report. Online: http://www.mcafee.com/hk/resources/reports/rp-quarterly-threat-q4-2014.pdf.

Milosevic, J., Dittrich, A., Ferrante, A., and Malek, M. (2014). A resource-optimized approach to efficient early detection of mobile malware. In *Availability, Reliability and Security (ARES), 2014 Ninth International Conference on*, pages 333–340. IEEE.

Milosevic, J., Ferrante, A., and Malek, M. (2016). What does the memory say? towards the most indicative features for efficient malware detection. In *CCNC 2016, The 13th Annual IEEE Consumer Communications & Networking Conference*, Las Vegas, NV, USA. IEEE Communication Society, IEEE Communication Society.

Moser, A., Kruegel, C., and Kirda, E. (2007). Limits of static analysis for malware detection. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 421–430.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., and Weiss, Y. (2012). "andromaly": A behavioral malware detection framework for android devices. *J. Intell. Inf. Syst.*, 38(1):161–190.

Symantec Corporation (2015). Internet security threat report volume 20. Technical report. Online: https://www.symantec.com/content/en/us/enterprise/other_resources/21347933_GA_RPT-internet-security-threat-report-volume-20-2015.pdf.

Truong, H. T. T., Lagerspetz, E., Nurmi, P., Oliner, A. J., Tarkoma, S., Asokan, N., and Bhattacharya, S. (2013). The Company You Keep: Mobile Malware Infection Rates and Inexpensive Risk Indicators. *CoRR*, abs/1312.3245.

Wu, D.-J., Mao, C.-H., Wei, T.-E., Lee, H.-M., and Wu, K.-P. (2012). Droidmat: Android malware detection through manifest and api calls tracing. In *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*, pages 62–69.

Zhou, Y. and Jiang, X. (2012). Dissecting Android Malware: Characterization and Evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP ’12, pages 95–109, Washington, DC, USA. IEEE Computer Society.