Applying Behavioral Detection on Android-Based Devices

Asaf Shabtai and Yuval Elovici

Deutsche Telekom Laboratories at Ben-Gurion University, and
Department of Information Systems Engineering
Ben-Gurion University, Be'er Sheva, 84105 Israel
{shabtaia,elovici}@bgu.ac.il

Abstract. We present Andromaly - a behavioral-based detection framework for Android-powered mobile devices. The proposed framework realizes a Host-based Intrusion Detection System (HIDS) that continuously monitors various features and events obtained from the mobile device, and then applies Machine Learning methods to classify the collected data as normal (benign) or abnormal (malicious). Since no malicious applications are yet available for Android, we evaluated Andromaly's ability to differentiate between game and tool applications. Successful differentiation between games and tools is expected to provide a positive indication about the ability of such methods to learn and model the behavior of an Android application and potentially detect malicious applications. Several combinations of classification algorithms, feature selections and the number of top features were evaluated. Empirical results suggest that the proposed detection framework is effective in detecting types of applications having similar behavior, which is an indication for the ability to detect unknown malware in the Android framework.

Keywords: Intrusion Detection, Mobile Devices, Machine Learning, Malware, Security, Android.

1 Introduction

Personal Digital Assistants (PDAs), mobile phones and recently smartphones have evolved from simple mobile phones into sophisticated yet compact minicomputers which can connect to a wide spectrum of networks, including the Internet and corporate intranets. Designed as open, programmable, networked devices, smartphones are susceptible to various malware threats such as viruses, Trojans, and worms, all of which are well-known from desktop platforms. These devices enable users to access and browse the Internet, receive and send emails, SMSs, and MMSs, connect to other devices for exchanging information/synchronizing, and activate various applications, which make these devices attack targets [1], [2].

A compromised smartphone can inflict severe damages to both users and the cellular service provider. Malware on a smartphone can make the phone partially or fully unusable; cause unwanted billing; steal private information (possibly by Phishing and Social Engineering); or infect every name in a user's phonebook [3].

The challenges for smartphone security are becoming very similar to those that personal computers encounter [5]. Common desktop-security solutions are being developed for mobile devices. Botha et al. [6] analyze common desktop security solutions and assess their applicability to mobile devices. Nevertheless, some of these solutions (such as antivirus software) are inadequate for use on smartphones as they consume too much CPU and memory and might result in rapid draining of the power source. In addition, most antivirus detection capabilities depend on the existence of an updated malware signature repository, therefore the antivirus users are not protected whenever an attacker spreads a previously un-encountered malware. Since the response time of antivirus vendors may vary between several hours to several days to identify the new malware, generate a signature, and update their clients' signature database, hackers have a substantial window of opportunity [7]. Some malware instances may target a specific and relatively small number of mobile devices (e.g., for extracted confidential information or track owner's location) and will therefore take quite a time till they are discovered.

In this research we describe a framework for detecting malware on Android mobile devices in the form of a Host-based Intrusion Detection System (HIDS). This is accomplished by continuously monitoring mobile devices to detect suspicious and abnormal activities. The framework relies on a light-weight agent that samples various system metrics, analysis of the sampled measurements and inference about the state of the device. The main assumption is that system metrics such as CPU consumption, number of sent packets through the Wi-Fi, number of running processes, battery level etc. can be employed for detection of previously un-encountered malware by examining similarities with patterns of system metrics induced by known malware [4], [8]. The primary goal of the study is to find the optimal mix of: a classification method, feature selection method and the number of monitored features that yields the best performance in accurately detecting new malware on Android. Since no malicious applications are yet available for Android, we evaluated Andromaly's ability to differentiate between game and tool applications.

2 Related Work

Our overview of related academic literature indicates that most extant research on protection of mobile devices has focused on applying and evaluating HIDS. These systems, using anomaly- or rule-based detection methods, extract and analyze (either locally or by a remote server) a set of features indicating the state of the device. Several systems are reviewed in this section.

Artificial Neural Networks (ANNs) were used in order to detect anomalous behavior indicating a fraudulent use of the operator services (e.g., registration with a false identity and using the phone to high tariff destinations) [9]. The Intrusion Detection Architecture for Mobile Networks (IDAMN) system [10] offers three levels of detection: location-based detection (a user active in two different locations at the same time); traffic anomaly detection (an area having normally low network activity, suddenly experiencing high network activity); and detecting anomalous behavior of individual mobile-phone users. Yap et al. [11] employ a behavior checker solution that detects malicious activities in a mobile system. They present a proof-of-concept scenario using a Nokia Mobile phone running a Symbian OS. In the demonstration, a behavioral detector detects a simulated Trojan attempting to use the message server component without authorization to create an SMS message. Cheng et al. [4] present SmartSiren, a collaborative proxy-based virus detection and alert system for smartphones. Single-device and system-wide abnormal behaviors are detected by the joint analysis of communication activity of monitored smartphones. Schmidt et al. [12] monitored a smartphone running a Symbian OS in order to extract features that describe the state of the device and which can be used for anomaly detection. These features were collected by a Symbian monitoring client and forwarded to a Remote Anomaly Detection System (RADS). The gathered data were used for anomaly detection methods in order to distinguish between normal and abnormal behavior.

An interesting behavioral detection framework is proposed [13] to detect mobile worms, viruses and Trojan horses. The method employs a temporal logic approach to detect malicious activity over time. An efficient representation of malware behaviors is proposed based on a key observation that the logical ordering of an application's actions over time often reveals malicious intent even when each action alone may appear harmless.

Special effort has been devoted to Intrusion Detection Systems (IDS) that analyze generic battery power consumption patterns to block Distributed Denial of Service (DDoS) attacks or to detect malicious activity via power depletion. Kim et al. [14] presented a power-aware, malware-detection framework that monitors, detects, and analyzes previously unknown energy-depletion threats. Buennemeyer et al. [15] introduced capabilities developed for a Battery-Sensing Intrusion Protection System (B-SIPS) for mobile computers, which alerts when abnormal current changes are detected. Nash et al. [16] presented a design for an intrusion detection system that focuses on the performance, energy, and memory constraints of mobile computing devices. Jacoby and Davis [17] presented a host Battery-Based Intrusion Detection System (B-BID) as a mean of improving mobile device security. The basic idea is that monitoring the device's electrical current and evaluating its correlation with known signatures and patterns, can facilitate attack detection and even identification.

Hwang et al. [18] evaluated the effectiveness of Keystroke Dynamics-based Authentication (KDA) on mobile devices. Their empirical evaluation focused on short PIN numbers (four digits) and the proposed method yielded a 4% misclassification rate.

The aforementioned frameworks and systems proved valuable in protecting mobile devices in general however, they do not leverage Android's capabilities to their full extent. Since Android is an open source and extensible platform it allows to extract as much features as we would like. This enables to provide richer detection capabilities, not relying merely on the standard call records [9], or power consumption patterns [15]-[18].

3 Anomaly Detection Framework for Android

Google's Android is a comprehensive software framework targeted towards such smart mobile devices (i.e., smartphones, PDAs), and it includes an operating system, a middleware and a set of key applications. Android emerged as an open-source, community-based framework which provides APIs to most of the software and hardware components. Specifically, it allows third-party developers to develop their own applications. The applications are written in the Java programming language based on the APIs provided by the Android Software Development Kit (SDK), but developers can also develop and modify kernel-based functionalities, which is not common for smartphone platforms.

We developed a lightweight Host-based Intrusion Detection System (in terms of CPU, memory and battery consumption) for Android-based mobile devices. The basis of the intrusion detection process consists of real-time, monitoring, collection, preprocessing and analysis of various system metrics, such as CPU consumption, number of sent packets through the Wi-Fi, number of running processes and battery level. System and usage parameters, changed as a result of specific events, may also be collected (e.g., keyboard/touch-screen pressing, application start-up). After collection and preprocessing, the system metrics are sent to analysis by various detection units, namely processors, each employing its own expertise to detect malicious behavior and generate a threat assessment (TA) accordingly. The pending threat assessments are weighted to produce an integrated alert and also includes a smoothing phase (combining the generated alert with the past history of alerts) in order to avoid instantaneous false alarms. After the weighting phase, a proper notification is displayed to the user. Moreover, the alert is matched against a set of automatic or manual actions that can be undertaken to mitigate the threat. Automatic actions include among others: uninstalling an application, killing of a process, disconnecting of all radios, encrypting data, changing firewall policies and more. A manual action can be uninstalling an application subject to user consent.

The components of the agent are clustered into four main groups (see Figure 1): Feature extraction, processors, agent service, and the Graphical User Interface (GUI). The *Feature Extractors* communicate with various components of the Android framework, including the Linux kernel and the Application Framework layer in order to collect feature metrics, while the Feature Manager triggers the Feature Extractors and requests new feature measurements every pre-defined time interval. In addition, the *Feature Manager* may apply some pre-processing on the raw features that are collected by the Feature Extractors.

A *Processor* is an analysis and detection unit. It is preferred that the processor will be provided as pluggable external component which can be seamlessly installed and un-installed. Its role is to receive feature vectors from the Main Agent Service, analyze them and output threat assessments to the Threat Weighting Unit. Each processor may expose an advanced configuration screen. Processors can be either: rule-based, knowledge-based, or classifiers/anomaly detector based on Machine Learning (ML) methods.

The *Threat Weighting Unit* (TWU) obtains the analysis results from all active processors and applies an ensemble algorithm (such as Majority Voting, Distribution Summation etc.) in order to derive a final coherent decision regarding a device's infection level. The Alert Manager receives the final ranking as produced by the TWU. It can then apply some smoothing function in order to provide a more persistent alert and to avoid instantaneous false alarms. Examples of such functions can be moving average and leaky-bucket. The smoothed infection level is then compared with pre-defined minimum and maximum thresholds.

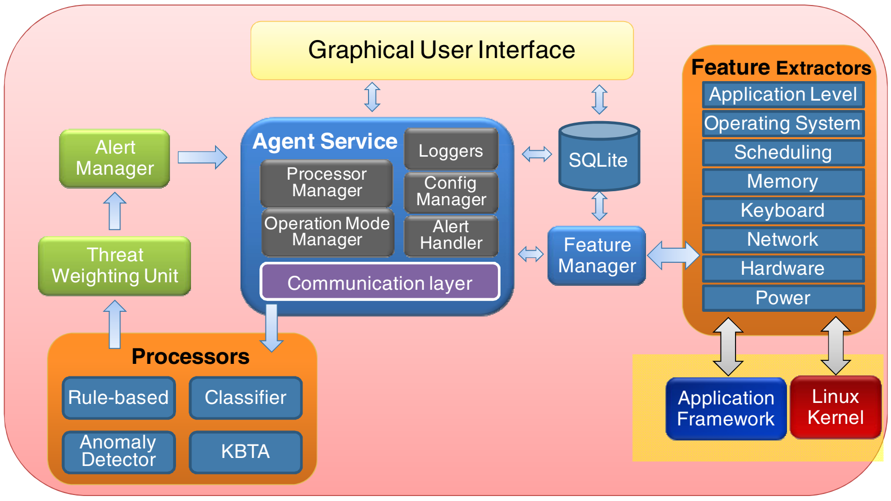

Fig. 1. The Andromaly Framework

The *Main Agent Service* is the most important component. This service synchronizes feature collection, malware detection and alert process. The Agent Service manages the detection flow by requesting new samples of features, sending newly sampled metrics to the processors and receiving the final recommendation from the *Alert Manager*. The *Loggers* provide logging options for debugging, calibration and experimentation with detection algorithms. The *Configuration Manager* manages the configuration of the agent (such as active processors, active feature extractors, alert threshold, active loggers, sampling temporal interval, detection mode configuration, etc.). The *Alert Handler* triggers an action as a result of a dispatched alert (e.g., visual alert in the notification bar, uninstalling an application sending notification via SMS or email, locking the device, disconnecting any communication channels). The *Processor Manager* registers/unregisters processors, and activates/deactivates processors. The Operation Mode Manger changes the agent from one operation mode to another based on the desired configuration. This will activate/deactivate processors and feature extractors. Changing from one operation mode to another (i.e. from Full Security mode to Normal mode) is triggered as a result of changes in available resources levels (battery, CPU, Network).

The last component is the *Graphical User Interface* which provides the user with the means to configure agent's parameters, activate/deactivate (for experimental usage only), visual alerting, and visual exploration of collected data.

4 Detection Method

4.1 Using Machine Learning for Behavioral Analysis

The evaluation of Machine Learning classifiers is typically split into two subsequent phases: training and testing. In the first phase, a training-set of games and tools feature vectors is provided to the system. These feature vectors are collected during the activation of both game and tool applications. The representative feature vectors in the training set and the real class of each vector (as game/tool) are assumed to be known and enable to calibrate the detection algorithms (such as a Decision Trees, or Bayesian Network). By processing these vectors, the algorithm generates a trained classifier.

Next, during the testing phase, a different collection (the testing-set) containing both game and tool applications feature vectors is classified by the trained classifier. In the testing phase, the performance of the classifier is evaluated by extracting standard accuracy measures for classifiers. Thus, it is necessary to know the real class of the feature vectors in the test-set in order to compare it real class with the class that was derived by the trained classifier.

Based on previous experience and after weighing the resource consumption issue, we decided to evaluate the following candidate classifiers: k-Means [19], Logistic Regression [20], Histograms [21], Decision Tree [22], Bayesian Networks [23] and Naïve Bayes [24].

4.2 Feature Selection

In Machine Learning applications, a large number of extracted features, some of which redundant or irrelevant, present several problems such as - misleading the learning algorithm, over-fitting, reducing generality, and increasing model complexity and run-time. These adverse effects are even more crucial when applying Machine Learning methods on mobile devices, since they are often restricted by processing and storage-capabilities, as well as battery power. Applying fine feature selection in a preparatory stage enabled to use our malware detector more efficiently, with a faster detection cycle. Nevertheless, reducing the amount of features should be performed while preserving a high level of accuracy.

Three feature selection methods were applied to the datasets: Information Gain (IG), Chi-Square (CS) and Fisher Score (FS). These feature selection methods follow the Feature Ranking approach and, using a specific metric, compute and return a score for each feature individually. Chi-Square [25] measures the lack of independence between a feature f and a class C. The Fisher Score [26] expresses the difference between two classes relative to a specific feature taking into account the mean and standard deviation of the feature's values in different classes. If the absolute difference between the feature's mean values in the two classes is small, and the sum of the feature's standard deviations of the two classes is large, the feature is not considered discriminative. Information Gain [27] determines the amount of information which a feature provides about a class by measuring how well it separates the training examples according to their target classification. In a more formal definition, Information Gain quantifies the expected reduction of Shannon's Entropy [27] caused by partitioning the examples according to a selected feature.

A problem was raised when we had to decide how many features we would choose for the classification task from the feature selection algorithms' output ranked lists. In order to avoid any bias by selecting an arbitrary number of features, we used, for each feature selection algorithm, three different configurations: 10, 20 and 50 features that were ranked the highest out of the 88 features ranked by the feature selection algorithms.

5 Evaluation

In order to evaluate our behavioral detection framework we performed two experiments. The research questions that we attempt to answer using the experiments are described in subsection 5.1. In subsection 5.2 we describe the dataset created for the experiments. Finally, subsection 5.3 describes the scheme of the two experiments and the obtained results.

5.1 Research Question

We evaluated the capability of the proposed HIDS framework to classify applications through two experiments, aimed at answering the following questions:

1) Is it possible to detect unknown instances of known application types on Android devices?
2) Which classifier is most accurate in detecting malware on Android devices: Decision Tree (DT), Naïve Bayes (NB), Bayesian Networks (BN), k-Means, Histogram or Logistic Regression (LR)?
3) Which number of extracted features and feature selection method yield the most accurate detection results: 10, 20 or 50 top- features selected using Chi-Square, Fisher Score or InfoGain?
4) What are the specific features that yield the maximum detection accuracy?

In order to perform the comparison between the various detection algorithms and feature selection schemes, we employed the following standard metrics: the True Positive Rate (TPR) measure, which is the proportion of positive instances classified correctly; False Positive Rate (FPR), which is the proportion of negative instances misclassified; and the Total Accuracy, which measures the proportion of absolutely correctly classified instances, either positive or negative.

5.2 Creating the Dataset for the Experiments

Since no standard dataset was available for this study, we had to create our own datasets. For the experiments, 23 games and 20 tools were collected, 11 of them were available on the Android framework, while the rest were obtained from the Android Market (Appendix A). All games and tools were verified to be virus-free before installation by manually exploring the permissions that the applications required, and by using a static analysis of dex files.

The aforementioned applications (i.e., 23 games, and 20 tools) were installed on five Android devices. The five devices were similar in the platform (HTC G1) with the same firmware and software versions. The five devices were used regularly by different users and thus varied in the amount and type of applications installed as well as usage patterns. The HIDS application, which continuously sampled various features on the device, was installed and activated on the devices under regulated conditions, and measurements were logged on the SD-card.

Each of the five Android devices had one user who used each of the 43 applications for 10 minutes, while in the background the malware detection system collected new feature vectors every 2 seconds. Therefore, a total of approximately 300 feature vectors were collected per each application and device. All the vectors in the datasets were labeled with their true class: 'game' or 'tool'. The table in Appendix A presents the number of vectors collected for each malicious, tool and game applications on the two tested devices.

The extracted features are clustered into two primary categories: Application Framework and Linux Kernel. Features belonging to groups such as Messaging, Phone Calls and Applications belong to the Application Framework category and were extracted through APIs provided by the framework, whereas features belonging to groups such as Keyboard, Touch Screen, Scheduling and Memory belong to the Linux Kernel category. A total of 88 features were collected for each monitored application (see Appendix B).

5.3 Experiments and Results

The purpose of the experiments was to evaluate the ability of the proposed detection methods to distinguish between games and tools applications. The following two experiments examine the performance of the detection system in different situations. For each experiment we used datasets extracted from 5 different devices, on which we evaluated 6 detection algorithms, 3 feature selection methods, and 3 sizes of top features groups (10, 20 and 50) as presented in Table 1.

Table 1. Experiments descriptions

Exp.	# of detection algorithms	# of feature selection methods	# of devices	# of top feature groups	# of iterations	Total number or funs	Testing on applications not in training set
I	6	3	5	3	20	5,400	-
II	6	3	5	3	20	5,400	+

Experiment 1

The purpose of this experiment is to evaluate the ability of each combination of detection algorithm, feature selection method, and number of top features to differentiate between game and tool applications when training set includes all game/tool applications. The training set contained 80% of the feature vectors of both the game and tool applications. The testing set contained the rest 20% feature vectors of the same game and tool applications (Figure 2a). The feature vectors were assigned in a random fashion. This experiment was repeated for each device 20 times, with different allocations of the training and testing set which results in a total of 5,400 runs (Table 1).

Experiment 2

The purpose of this experiment is to evaluate the ability of each combination of detection algorithm, feature selection method, and number of top features to differentiate between game and tool applications not included in the training set. The configuration of this experiment resembles the first one. However, unlike the first experiment the training set contained feature vectors clusters for 80% of all games and 80% of all tools. The testing set contained feature vectors clusters of the rest of the 20% games and 20% tools that were not included in the training set on the same device

(Figure 2b). This examined the ability of the different algorithms to detect unknown applications. This experiment was repeated for each device 20 times, with different allocations of the training and testing set which results in a total of 5,400 runs (Table 1).

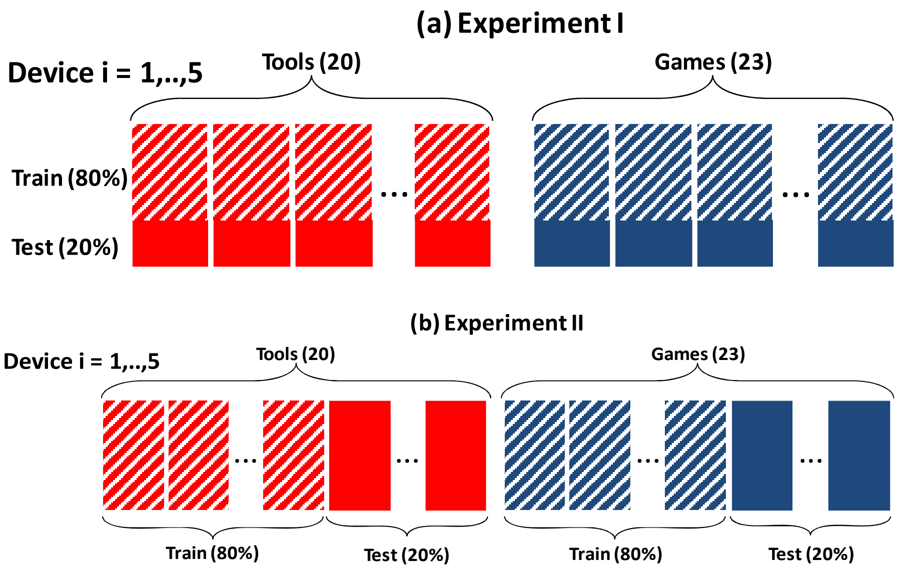

Fig. 2. Illustration of the datasets in each experimental configuration

Figure 3 presents, for each experiment, the average Accuracy and FPR, of the five devices when combined together, for each detector. The results show that the best detectors (classifiers) in experiment I and II are Decision Tree, Logistic Regression and Bayesian Networks. Additionally, we observed that in experiment I the five devices had similar results for BN, DT, Logistic Regression and NB. For experiment II the five devices had similar results for Logistic Regression and NB.

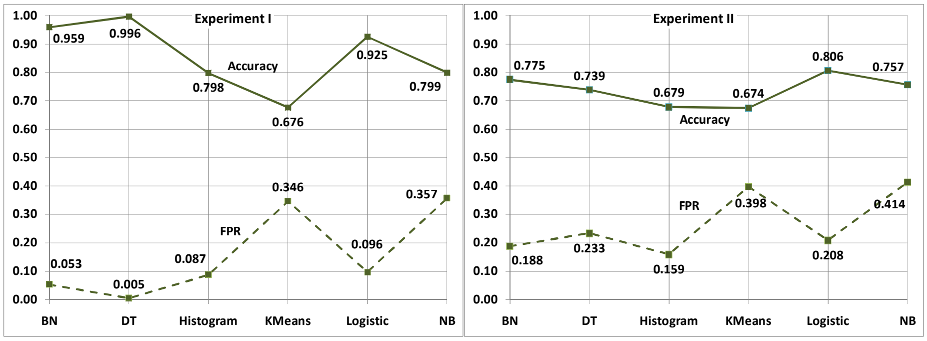

Fig. 3. Average Accuracy and FPR for each one of the detectors

Table 2 presents the averaged Accuracy and FPR of the three feature selection methods and the three top numbers of selected features. The results indicate that for experiment I, InfoGain with top 20 features outperformed all the other combinations. For experiment II Fisher Score with top 10 outperformed all the other combinations.

Table 2. Averaged Accuracy and FPR for each one of the feature selection methods and top features

Exp	Feature Selection Method	FPR			Accuracy		
		10	20	50	10	20	50
I	ChiSquare	0.160	0.134	0.172	0.852	0.876	0.850
	FisherScore	0.152	0.174	0.167	0.857	0.860	0.857
	InfoGain	0.155	**0.129**	0.174	0.850	**0.877**	0.850
II	ChiSquare	0.270	0.258	0.280	0.732	0.751	0.725
	FisherScore	**0.250**	0.263	0.268	**0.750**	0.750	0.735
	InfoGain	0.265	0.263	0.282	0.729	0.747	0.724

Figure 4 present the selected features for each feature selection method. The top 10 selected features for each feature selection method were picked as follows. All of the feature selection methods ranked each feature that they pick. A higher rank indicates that the feature differentiates better between game and tool applications using the feature selection method. For each device we listed the top 10 features in a descending order according to their rank for each feature selection method. Corresponding to the features' rank, each feature was given a score starting from 10 to 1 (10 for the most significant feature according to the feature selection method). Then, for each feature selection method we calculated the sum of scores over all the devices for each feature selection method. Additionally, we omitted features with a low score and features that were chosen only for one device. The features in the graph are ordered primarily by the number of devices that selected the feature and then by their score. From Figure 4 we conclude that Chi-Square and InfoGain graded the same top 10 selected features with a very similar rank. Both of them assigned the highest ranks to the following features: Load_Avg_15_mins, Total_Entities, Running_Processes, Mapped_Pages, Battery_Voltage, Context_Switches, Avg_Key_Dwell_Time, Schedule_Calls and Anonymous_Pages.

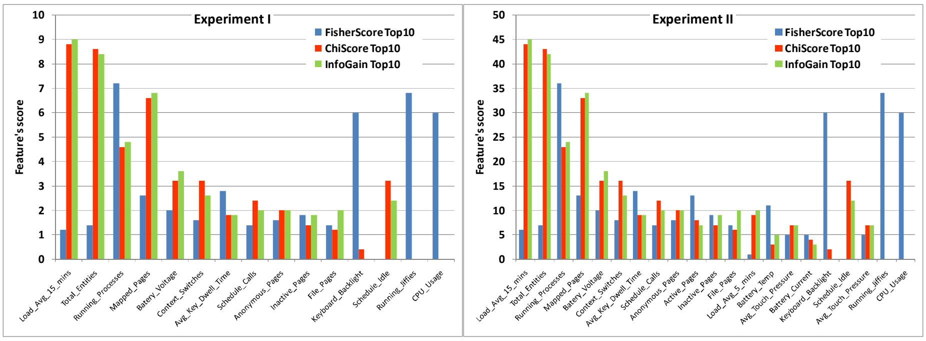

Fig. 4. Best selected features

6 Discussion and Conclusions

We presented an HIDS framework for Android which employs Machine Learning and tested various feature selection methods and classification/anomaly detection algorithms. The detection approach and algorithms are light-weight and run on the device itself. There is however also an option to perform the detection at a centralized location, or at least report the analysis results such a location. This can be useful in detection of malware propagation patterns across a community of mobile devices. As the framework is open, it is also possible to accommodate additional malware detection techniques (e.g., knowledge-based inferences [28]).

Table 3 presents the best configurations, which outperformed all other configurations in terms of ability to differentiate between game and tool applications for each of the experiments (averaged over the five devices and all iterations).

Table 3. Averaged Accuracy, FPR, TPR and AUC of the best configuration in each experiment

Experiment	Best Configuration	TRP	FPR	AUC	Accuracy
I	DT\J48 InfoGain 20	0.997	0.004	0.998	0.997
II	LR FisherScore 20	0.828	0.199	0.888	0.818

From the results we conclude that anomaly detection is feasible on Android devices. The successful differentiation between games and tools provides a positive indication about the ability of such methods to learn and model the behavior of an Android application and potentially detect malicious applications. Furthermore, the fact that the detection can be effective even when using a small number of features (20 features were sufficient in both experiments) and simple detection algorithms ensure that stringent resource constraints (i.e., CPU, battery) on the device are met. These findings are congruent with earlier work which noted that most of the top ten applications preferred by mobile phone users affected the monitored features in different patterns [12]. This observation strengthens the viability of anomaly detection techniques for detection malware on mobile devices.

Looking at the performance of detectors on each of the five devices separately, it also is evident that they exhibit similar performance. This supports the external validity of our experiments by indicating that the selected features in our experiments are not user-, or configuration-dependent, and that we can learn on a set of devices and detect effectively even on other devices. Additionally, there is high similarity in the features that were selected on each of the experiments. We suggest that it is due to a unique and persistent behavior of applications across devices.

Several avenues can be explored for future research. First and foremost we would like to understand whether we can train the classifiers on a set of devices and can still detect effectively on other devices. Second, we can alert about a detected anomaly when it persists. Third, we can add a temporal perspective by augmenting the collected features with a time stamp (e.g., use the change of the battery level in the last 10min rather than the level of the battery at a certain point in time), or logging sequences of events (e.g., a Boolean feature that is true if there was an access to an SD-card concurrently with a high volume of network traffic via Wi-Fi). Finally, we can focus on monitoring and detection of malicious processes rather than monitoring the whole system. This will enable to isolate the activities of specific applications.

References

1. Leavitt, N.: Mobile phones: the next frontier for hackers? Computer 38(4), 20–23 (2005)
2. Shih, D.H., Lin, B., Chiang, H.S., Shih, M.H.: Security aspects of mobile phone virus: a critical survey. Industrial Management & Data Systems 108(4), 478–494 (2008)
3. Piercy, M.: Embedded devices next on the virus target list. IEEE Electronics Systems and Software 2, 42–43 (2004)
4. Cheng, J., Wong, S.H., Yang, H., Lu, S.: SmartSiren: virus detection and alert for smartphones. In: Proceedings of the 5th International Conference on Mobile Systems, Applications and Services (2007)
5. Muthukumaran, D., et al.: Measuring integrity on mobile phone systems. In: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (2008)
6. Botha, R.A., Furnell, S.M., Clarke, N.L.: From desktop to mobile: Examining the security experience. Computer & Security 28, 130–137 (2009)
7. Dagon, C., Martin, T., Starner, T.: Mobile phones as computing devices the viruses are coming. Pervasive Computing, 11–15 (2004)
8. Emm, D.: Lasco: the hybrid threat. Computer Fraud and Security (2005)
9. Moreau, Y., Preneel, B., Burge, P., Shawe-Taylor, J., Stoermann, C., Cooke, C.: Novel techniques for fraud detection in mobile telecommunication networks. In: ACTS Mobile Summit (1997)
10. Samfat, D., Molva, R.: IDAMN: An intrusion detection architecture for mobile networks. IEEE Journal on Selected Areas in Communications 15(7), 1373–1380 (1997)
11. Yap, T.S., Ewe, H.T.: A mobile phone malicious software detection model with behavior checker. In: Shimojo, S., Ichii, S., Ling, T.-W., Song, K.-H. (eds.) HSI 2005. LNCS, vol. 3597, pp. 57–65. Springer, Heidelberg (2005)
12. Schmidt, A., Peters, F., Lamour, F., Albayrak, S.: Monitoring smartphones for anomaly detection. In: Proceedings of the 1st International Conference on Mobile Wireless Middleware,Operating Systems, and Applications (2008)
13. Bose, A., Hu, X., Shin, K.G., Park, T.: Behavioral detection of malware on mobile handsets. In: Proceeding of the 6th International Conference on Mobile Systems, Applications, and Services (2008)
14. Kim, H., Smith, J., Shin, K.G.: Detecting energy-greedy anomalies and mobile malware variants. In: Proceeding of the 6th International Conference on Mobile Systems, Applications, and Services (2008)
15. Buennemeyer, T.K., et al.: Mobile device profiling and intrusion detection using smart batteries. In: International Conference on System Sciences, pp. 296–296 (2008)
16. Nash, D.C., et al.: Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices. In: Pervasive Computing and Communications Workshops (2005)
17. Jacoby, G.A., Davis, N.J.: Battery-based intrusion detection. In: Global Telecommunications Conference (2004)
18. Hwang, S.S., Cho, S., Park, S.: Keystroke dynamics-based authentication for mobile Devices Computer & Security 28, 85–93 (2009)
19. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering. ACM Computing Surveys 31(3), 264–296 (1999)
20. Neter, J.K., Nachtsheim, M.H., Wasserman, W.: Applied Linear Statistical Models. McGraw-Hill, New York (1996)
21. Endler, D.: Intrusion detection: Applying machine learning to solaris audit data. In: Proceedings of the 14th Annual Computer Security Applications Conference (1998)

22. Quinlan, J.R.: C4.5: Programs for machine learning. Morgan Kaufmann Publishers, Inc., San Francisco (1993)
23. Pearl, J.: Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kauhmann, San Francisco (1988)
24. Russel, S., Norvig, P.: Artificial Intelligence: A modern approach. Prentice Hall, Englewood Cliffs (2002)
25. Imam, I.F., Michalski, R.S., Kerschberg, L.: Discovering Attribute Dependence in Databases by Integrating Symbolic Learning and Statistical Analysis Techniques. In: Proceeding of the AAAI 1993 Workshop on Knowledge Discovery in Databases (1993)
26. Golub, T., et al.: Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. Science 286, 531–537 (1999)
27. Shannon, C.E.: The mathematical theory of communication. The Bell system Technical Journal 27(3), 379–423 (1948)
28. Shabtai, A., Kanonov, U., Elovici, Y.: Intrusion Detection on Mobile Devices Using the Knowledge Based Temporal-Abstraction Method. Journal of Systems and Software, (2010) doi:10.1016/j.jss.2010.03.046

Appendix A - List of Used Applications

	Application	Device1	Device2	Device3	Device4	Device5
Games	abduction	322	197	205	257	341
	armageddonoid	343	222	198	264	326
	battleformars	544	208	269	294	582
	bonsai	355	287	267	253	304
	breadfactory	300	208	241	246	291
	connect4	308	204	225	249	289
	flyinghigh	279	250	206	234	365
	froggo	308	185	241	263	301
	hangemhigh	342	348	274	255	339
	labyrinthlite	317	297	244	252	324
	lexic	354	266	222	273	347
	minesweeper	342	267	249	285	541
	mushroom	251	172	193	199	209
	pairup	410	275	298	259	582
	picacrossexpress	300	357	255	302	332
	smarttactoe	298	233	254	262	300
	snake	294	254	250	233	314
	solitaire	334	378	281	288	439
	switcher	303	231	269	236	395
	tankace	336	287	250	262	330
	throttlecopter	321	230	228	210	280
	trap	454	245	189	251	398
	wordpops	301	314	302	320	267
Tools	browser	307	274	218	336	300
	calculator	296	251	266	276	365
	calendar	319	233	250	270	341
	camera	302	251	249	260	294
	contacts	303	230	218	256	680
	email	219	250	445	275	339
	im	371	245	238	251	385
	iofilemanager	295	235	241	289	284
	maps	342	216	245	278	429
	messaging	322	247	255	278	296
	music	304	251	272	256	343
	mytracks	356	233	269	271	543
	noteeverything	329	212	287	275	408
	oxforddictionary	323	275	272	283	374
	pdfviewer	280	249	248	263	319
	phonalyzer	304	240	268	290	318
	phone	125	224	140	110	244
	tasks	300	226	250	263	302
	voicememo	312	230	270	253	269
	weather	372	242	272	269	297

Appendix B - List of Collected Features

Collected Features (88)		
Touch screen:	**Memory:**	**Network:**
Avg_Touch_Pressure	Garbage_Collections	Local_TX_Packets
Avg_Touch_Area	Free_Pages	Local_TX_Bytes
Keyboard:	Inactive_Pages	Local_RX_Packets
Avg_Key_Flight_Time	Active_Pages	Local_RX_Bytes
Del_Key_Use_Rate	Anonymous_Pages	WiFi_TX_Packets
Avg_Trans_To_U	Mapped_Pages	WiFi_TX_Bytes
Avg_Trans_L_To_R	File_Pages	WiFi_RX_Packets
Avg_Trans_R_To_L	Dirty_Pages	WiFi_RX_Bytes
Avg_Key_Dwell_Time	Writeback_Pages	**Hardware:**
Keyboard_Opening	DMA_Allocations	Camera
Keyboard_Closing	Page_Frees	USB_State
Scheduler:	Page_Activations	**Binder:**
Yield_Calls	Page_Deactivations	BC_Transaction
Schedule_Calls	Minor_Page_Faults	BC_Reply
Schedule_Idle	**Application:**	BC_Acquire
Running_Jiffies	Package_Changing	BC_Release
Waiting_Jiffies	Package_Restarting	Binder_Active_Nodes
CPU Load:	Package_Addition	Binder_Total_Nodes
CPU_Usage	Package_Removal	Binder_Ref_Active
Load_Avg_1_min	Package_Restart	Binder_Ref_Total
Load_Avg_5_mins	UID_Removal	Binder_Death_Active
Load_Avg_15_mins	**Calls:**	Binder_Death_Total
Runnable_Entities	Incoming_Calls	Binder_Transaction_Active
Total_Entities	Outgoing_Calls	Binder_Transaction_Total
Messaging:	Missed_Calls	Binder_Trns_Complete_Active
Outgoing_SMS	Outgoing_Non_CL_Calls	Binder_Trns_Complete_Total
Incoming_SMS	**Operating System:**	**Leds:**
Outgoing_Non_CL_SMS	Running_Processes	Button_Backlight
Power:	Context_Switches	Keyboard_Backlight
Charging_Enabled	Processes_Created	LCD_Backlight
Battery_Voltage	Orientation_Changing	Blue_Led
Battery_Current		Green_Led
Battery_Temp		Red_Led
Battery_Level_Change		
Battery_Level		