J Intell Inf Syst (2012) 38:161–190
DOI 10.1007/s10844-010-0148-x

"Andromaly": a behavioral malware detection framework for android devices

Asaf Shabtai · Uri Kanonov · Yuval Elovici · Chanan Glezer · Yael Weiss

Received: 22 August 2010 / Revised: 27 October 2010 / Accepted: 17 November 2010 /
Published online: 6 January 2011

Abstract This article presents Andromaly—a framework for detecting malware on Android mobile devices. The proposed framework realizes a Host-based Malware Detection System that continuously monitors various features and events obtained from the mobile device and then applies Machine Learning anomaly detectors to classify the collected data as normal (benign) or abnormal (malicious). Since no malicious applications are yet available for Android, we developed four malicious applications, and evaluated Andromaly's ability to detect new malware based on samples of known malware. We evaluated several combinations of anomaly detection algorithms, feature selection method and the number of top features in order to find the combination that yields the best performance in detecting new malware on Android. Empirical results suggest that the proposed framework is effective in detecting malware on mobile devices in general and on Android in particular.

Keywords 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

A. Shabtai (✉) · U. Kanonov · Y. Elovici · C. Glezer · Y. Weiss
Deutsche Telekom Laboratories at Ben-Gurion University, Department of Information Systems Engineering, Ben-Gurion University, Be'er Sheva 84105, Israel
e-mail: shabtaia@bgu.ac.il

U. Kanonov
e-mail: kanonov@bgu.ac.il

Y. Elovici
e-mail: elovici@bgu.ac.il

C. Glezer
e-mail: chanan@bgu.ac.il

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, Trojan horses, 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 (Leavitt 2005; Shih et al. 2008).

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 (Piercy 2004). Possible attack vectors into smartphones include: Cellular networks, Internet connections (via Wi-Fi, GPRS/EDGE or 3G network access); USB/ActiveSync/Docking and other peripherals (Cheng et al. 2007).

The challenges for smartphone security are becoming very similar to those that personal computers encounter (Muthukumaran et al. 2008) and common desktop-security solutions are often being downsized to mobile devices. As a case in point, (Botha et al. 2009) analyzed common desktop security solutions and evaluated their applicability to mobile devices. However, some of the desktop solutions (i.e., 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 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 (Dagon et al. 2004). 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 generic and modular framework for detecting malware on Android mobile devices. This is accomplished by continuously monitoring mobile devices to detect suspicious and abnormal activities using a supervised anomaly detection technique (Chandola et al. 2009). The framework relies on a light-weight application, installed on the mobile device that samples various system metrics and analyzes them in order to make inferences about the well-being 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 (Cheng et al. 2007; Emm 2006). This concept was initially presented by (Lee et al. 1999) who proposed a data mining framework that supports the task of intrusion detection. The central idea was to utilize auditing programs to extract an extensive set of features that describe each network connection or host session, and apply data mining methods to learn rules that capture the behavior of intrusions and normal activities. These rules can then be used for misuse detection and anomaly detection. New detection models are then incorporated into an existing IDS through a meta-

learning (or co-operative learning) process, which produces a meta-detection model that combines evidence from multiple models.

The primary goal of our study is to identify an 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.

2 Related work

The related work to this study in described in the following subsections. In Section 2.1 we present a review on general malware detection techniques followed by Section 2.2 that review malware detection techniques in mobile devices.

2.1 Malware detection techniques

Modern computer and communication infrastructures are highly susceptible to various types of attack. A common way of launching these attacks is by means of malicious software (malware) such as worms, viruses, and Trojan horses, which, when spread, can cause severe damage to private users, commercial companies and governments. The recent growth in high-speed Internet connections has led to an increase in the creation of new malware.

Several analysis techniques for detecting malware have been proposed. Basically static and dynamic analysis is distinguished. In *Dynamic Analysis* (also known as behavioral-based analysis) the detection consists of information that is collected from the operating system at runtime (i.e., during the execution of the program) such as system calls, network access and files and memory modifications (Rieck et al. 2008; Moskovitch et al. 2008). Systems such as *Panorama* (Yin et al. 2007) and the *BHO Spyware Taint Analyzer* (Egele et al. 2007) demonstrate the feasibility of dynamic methods by capturing suspicious information access patterns and processing behaviors (i.e., information flow analysis) that are common to different types of malware. Transforming these detection approaches to a mobile environment, however, is not straight forward due to resource constraints (i.e., CPU, power, memory) imposed by smartphones.

In *Static Analysis*, information about the program or its expected behavior consists of explicit and implicit observations in its binary/source code. While being fast and effective, static analysis techniques are limited, mainly due to the fact that various obfuscation techniques can be used to evade static analysis and thus render their ability to cope with polymorphic malware limited (Moser et al. 2007).

In the dynamic analysis approach the problems resulting from the various obfuscation methods do not exist, since the actual behavior of the program is monitored. However, this approach suffers from other disadvantages. First, it is difficult to simulate the appropriate conditions, in which the malicious functions of the program will be activated (such as the vulnerable application that the malware exploits). Second, it is not clear what is the required period of time needed to observe the appearance of the malicious activity for each malware.

Static and dynamic analysis solutions are primarily implemented using two methods: signature-based and heuristic-based. *Signature-based* is a common method used

by antivirus vendors and it relies on the identification of unique signatures that define the malware (Griffin et al. 2009). While being very precise, signature-based methods are useless against unknown malicious code. The *heuristic-based* methods are based on rules which are either determined by experts or by machine learning techniques that define a malicious or a benign behavior, in order to detect unknown malware (Jacob et al. 2008; Gryaznov 1999).

Recently, classification algorithms were employed to automate and extend the idea of heuristic-based methods. In these methods, the binary code of a file or the behavior of the application during execution is represented, and classifiers are used to learn patterns in order to classify new (unknown) applications as malicious or benign (Moskovitch et al. 2008; Shabtai et al. 2009c; Menahem et al. 2008).

2.2 Malware detection in mobile devices

Our overview of related academic literature indicates that most extant research on protection of mobile devices has focused on dynamic analysis approaches. In the majority of cases, these studies have proposed and evaluated Host-based Intrusion Detection Systems (HIDS). These systems (using anomaly- (Garcia-Teodoro et al. 2009) 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 at runtime. Several systems are reviewed in this section as summarized in Table 1. Only a handful of systems employ static analysis for detecting malware on mobile devices. Chaudhuri (2009) presented a formal language for describing Android applications and data flow among application components which was later incorporated in the ScanDroid (Adam et al. 2009).

Artificial Neural Networks (ANNs) were used in Moreau et al. (1997) 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). The *Intrusion Detection Architecture for Mobile Networks* (IDAMN) system (Samfat and Molva 1997) uses both rule-based and anomaly detection methods. IDAMN 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 and Ewe (2005) 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. The behavioral checker, however, cannot detect new attacks since it relies on specific definitions of malware behaviors. Cheng et al. (2007) 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. The evaluation of SmartSiren focused on detecting SMS viruses by monitoring the amount of SMSs send by a single device. Schmidt et al. (2009) 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

Table 1 Academic research on protection of mobile devices

Paper	Approach	Detection method	Detects
Moreau et al. (1997)	HIDS	Anomaly detection using ANN	Fraudulent use of the operator services such as high rate calls.
Samfat and Molva (1997) (IDAMN)	HIDS, NIDS	Anomaly detection; Rule-based detection	A user is being active in two different locations at the same time; traffic anomaly detection; and detecting anomalous behavior of individual mobile-phone users based on the telephone activity (such as call duration) and user's location in the network.
Yap and Ewe (2005)	HIDS	Signature-based detection	Proof of concept—detects unauthorized attempt to create SMS message.
Cheng et al. (2007) (SmartSiren)	HIDS, NIDS	Anomaly detection	Detects anomaly behavior of the device and outbreak of worm-related malware.
Schmidt et al. (2009)	HIDS	Anomaly detection	Monitor a smartphone running Symbian operating system and Windows Mobile in order to extract features for anomaly detection. These features are sent to a remote server for further analysis.
Schmidt et al. (2008)	HIDS	Anomaly detection	Monitor Linux events at OS and application level on an Android device. Event-based detection is demonstrated on the feature of static function calls.
Bose et al. (2008)	HIDS	Signature-based detection.	Using Temporal Logic to detect malicious activity over time that matches a set of signatures represented as a sequence of events.
Kim et al. (2008)	HIDS	Signature-based detection	Detects, and analyzes previously unknown energy-depletion threats based on a collection of power signatures.
Buennemeyer et al. (2008) (B-SIPS)	HIDS, NIDS	Anomaly detection	Detects abnormal current changes and its correlation with network attack.
Nash et al. (2005)	HIDS	Statistical method (linear regression)	Detects processes that are likely to be battery-draining attacks.
Jacoby and Davis (2004) (B-BID)	HIDS	Signature-based detection	Monitoring power consumption against "normal" power consumption range. Once an anomaly is detected, various system data is matched against known attack signatures.
Miettinen et al. (2006)	HIDS, NIDS	Event correlation	Combines both host-based and network-based data collection in order to be able to utilize the strengths of both detection approaches.
Hwang et al. (2009)	Authentication	KDA	Collected 5 features to train and build a classifier capable of detecting impostor users. Utilized artificial rhythms and tempo cues to overcome problems resulting from short PIN length.

gathered data can be used for anomaly detection methods in order to distinguish between normal and abnormal behavior; however, (Schmidt et al. 2008) demonstrated the ability of differentiating applications using the collected features and the framework was not tested for detecting abnormal behavior (i.e., malicious behavior). Additionally, the processing of the collected data was performed on a remote server, whereas in this paper we attempt to understand the feasibility of applying the detection on the device. Schmidt et al. (2008) focus on monitoring events at the kernel; that is, identifying critical kernel, log file, file system and network activity events, and devising efficient mechanisms to monitor them in a resource limited environment. They demonstrated their framework on static function call analysis and performed a statistical analysis on the function calls used by various applications.

An interesting behavioral detection framework is proposed in Bose et al. (2008) to detect mobile worms, viruses and Trojans. 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. The ability of this framework to detect new types of malware is still dubious as it requires a process of specifying temporal patterns for the malicious activities.

Special efforts were invested in research pertaining 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. For example, Kim et al. (2008) present a power-aware, malware-detection framework that monitors, detects, and analyzes previously unknown energy-depletion threats.

Buennemeyer et al. (2008) 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. (2005) presented a design for an intrusion detection system that focuses on the performance, energy, and memory constraints of mobile computing devices. Jacoby and Davis (2004) 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.

Miettinen et al. (2006) claim that host-based approaches are required, since network-based monitoring alone is not sufficient to encounter the future threats. They adopt a hybrid network/host-based approach. A correlation engine on the back-end server filters the received alarms according to correlation rules in its knowledge base and forwards the correlation results to a security monitoring GUI to be analyzed by security-monitoring administrators. Hwang et al. (2009) 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.

All in all, 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 many features as we would like. This enables to provide richer detection capabilities, not relying merely on the standard call records (Emm 2006),

or power consumption patterns (Kim et al. 2008; Buennemeyer et al. 2008; Nash et al. 2005; Jacoby and Davis 2004). Our research is innovative in that is evaluates the ability to detect malicious activity on an Android device, by employing Machine Learning algorithms using a variety of monitored features. To ensure scalability and maintain low resource consumption of the detection process, we used simple, standard, Machine Learning algorithms.

3 The proposed andromaly framework

Google's Android[1] 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.

The security model of Android (and that of many other phone operating systems) is "system centric" (i.e., the system focuses on protecting itself). Applications statically identify the permissions that govern the rights to their data and interfaces at installation time. However, the application/developer has limited ability thereafter to govern to whom those rights are given or how they are later exercised (Ongtang et al. 2009).

In order to overcome this limitation we propose a lightweight Malware Detection System (in terms of CPU, memory and battery consumption) for Android-based mobile devices in order to assist users in detecting (and optionally reporting to the Android community) suspicious activities on their handsets. The basis of the malware 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/touchscreen 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. The weighting process is per threat type, i.e., virus TAs are weighted separately from worm TAs 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

[1] http://www.android.com

a process, disconnecting 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 proposed application are clustered into four main groups (see Fig. 1): Feature Extractors, Processors, Main 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 a pluggable external component which can be seamlessly installed and un-installed. Its role is to receive feature vectors from the *Main Service*, analyze them and output threat assessments to the *Threat Weighting Unit*. Each processor may expose an advanced configuration screen. Processors can be rule-based, knowledge-based, or classifiers/anomaly detectors employing Machine Learning 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.

The *Main Service* is the most important component. This service synchronizes feature collection, malware detection and alert process. The Main Service manages the

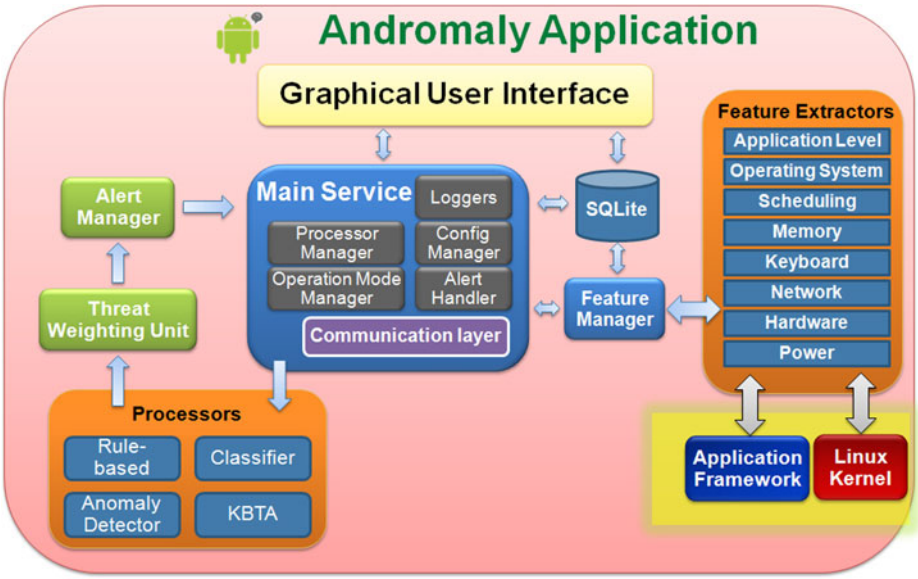

Fig. 1 The andromaly architecture

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 application (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 Manager* changes the application 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 application's parameters, activate/deactivate (for experimental usage only), visual alerting, and visual exploration of collected data.

Figure 2 presents a sequence diagram describing the flow of the main scenario of the Andromaly application. The process starts with the Main Service which calls the *collectData()* method each pre-defined temporal interval (t seconds) and passes the list of *MonitoredData* objects as returned by the Feature Manager to all active Processors (each *MonitoredData* object contains the name of the feature, value, measurement start-time and measurement end-time). The Main Service

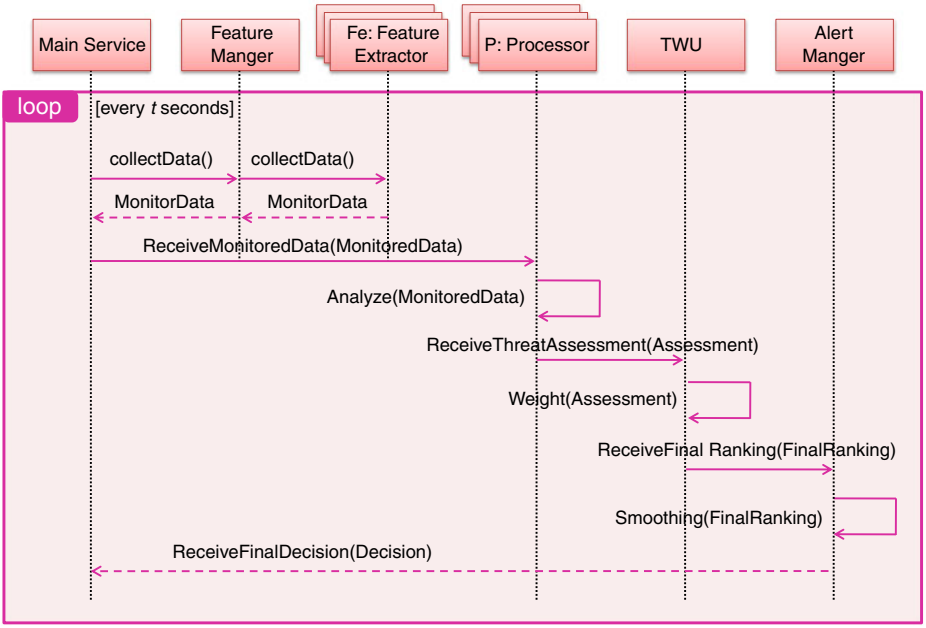

Fig. 2 Sequence diagram describing the flow of the main scenario of the malware detector

passes the list of *MonitoredData* objects by calling the Processors' *receiveMonitoredData(MonitoredData)* method. This is where the processing of the information occurs. Upon completion, a threat assessment is prepared by the processors and is dispatched to the Threat Weighting Unit (TWU). The TWU collects all of the processors' assessments and applies an ensemble algorithm to derive a coherent decision regarding the level of risk. The final ranking is further processed by the Alert Manager which informs the Main Service whether a malicious behavior was detected and what action to take.

4 Detection method

4.1 Using machine learning for malware detection

The proposed framework employs Machine Learning classification techniques for realizing a Malware Detection System. Under such an approach, the malware detector continuously monitors various features and events obtained from the system and then applies standard Machine Learning classifiers to classify collected observations as either *normal* (benign) or *abnormal* (malicious). Based on previous experience and after weighing the resource consumption issue, we decided to evaluate the following candidate classifiers: *k*-Means (Jain et al. 1999), Logistic Regression (Neter et al. 1996), Histograms (Endler 1998), Decision Tree (Quinlan 1993), Bayesian Networks (Pearl 1988) and Naïve Bayes (Russel and Norvig 2002; John and Langley 1995; Domingos and Pazzani 1997).

Evaluation of Machine Learning classifiers is typically split into two subsequent phases: training and testing. In the first phase, a training-set of benign and malicious feature vectors is provided to the system. These feature vectors are collected both during normal operation of the system (reflecting benign behavior) and when malware is activated (reflecting abnormal behavior of the system). The representative feature vectors in the training set and the real classification of each vector (benign/malicious) are assumed to be known to the learning algorithms and by processing these vectors, the algorithm generates a trained classifier. Next, during the testing phase, a different collection (the testing-set) containing both benign and malicious feature vectors is classified by the trained classifier. In the testing phase, the performance of the classifier is evaluated by extracting standard evaluation measures for classifiers. Thus, it is necessary to know the real class of the feature vectors in the test-set in order to compare the actual (i.e., real) class with the class generated by the trained classifier.

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.

We used a *Filter* approach for feature selection, due to its fast execution time and its generalization ability—as it isn't bound to the biases of any classifier (unlike the Wrapper approach) (Mitchell 1997). Under a *Filter* approach, an objective function evaluates features by their information content and estimates their expected contribution to the classification task. Three feature selection methods were applied to the datasets: *Chi-Square* (CS) (Imam et al. 1993), *Fisher Score* (FS) (Golub et al. 1999) and *Information Gain* (Shannon 1948). These feature selection methods follows the *Feature Ranking* approach and, using a specific metric, compute and return a score for each feature individually.

A problem was raised when we had to decide how many features to 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 featured ranked by the feature selection algorithms.

5 Evaluation

In order to evaluate our malware detection framework we performed several experiments. The research questions that we attempt to answer using the experiments are described in Section 5.1. In Section 5.2 we describe the dataset created for the experiments. Section 5.3 describes the scheme of the four experiments and Section 5.4 presents the obtained results. Finally, in Section 5.5 we discuss the impact of the malware detection framework on the handset resources.

5.1 Research questions

We evaluated the proposed framework through a set of four subsequent experiments, aimed at answering the following questions:

1. Is it possible to detect unknown malicious applications on Android devices using the Andromaly framework?
2. Is it possible to learn the behavior of applications on a set of Android devices and perform the detection on other devices?
3. Which classification algorithm is most accurate in detecting malware on Android devices: DT, NB, BN, *k*-Means, Histogram or Logistic Regression?
4. 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?
5. Which specific features yield maximum detection accuracy?

In order to perform the comparison between the various detection algorithms and feature selection schemes, we employed the following standard metrics: *True Positive Rate* (*TPR*) measure,[2] which is the proportion of positive instances (i.e.,

[2] Also known as Detection Rate in the intrusion detection community.

feature vectors of malicious applications) classified correctly (1); *False Positive Rate* (*FPR*),[3] which is the proportion of negative instances (i.e., feature vectors of benign applications) misclassified (2); and *Total Accuracy*, which measures the proportion of absolutely correctly classified instances, either positive or negative (3); where, *TP* is number of positive instances classified correctly; *FP* is the number of negative instances misclassified; *FN* is the number of positive instances misclassified; and *TN* is the number of negative instances classified correctly.

Additionally, we used the *Receiver Operating Characteristic* (*ROC*) curve and calculated the *Area-Under-Curve* (*AUC*). The ROC curve is a graphical representation of the trade-off between the TPR and FPR for every possible detection cut-off.

$$TPR = \frac{TP}{TP + FN} \tag{1}$$

$$FPR = \frac{FP}{FP + TN} \tag{2}$$

$$Total\ Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

5.2 Creating the dataset for the experiments

There are several types of threats targeting mobile malware. In our research we focus on attacks against the phones themselves and not the service provider's infrastructure (Guo et al. 2004). Four classes of attacks on a mobile device were identified (Botha et al. 2009): unsolicited information, theft-of-service, information theft and Denial-of-Service (DoS). Since Android is a relatively new platform and at the time we had performed our experiments there were no known instances of Android malware at our disposal, we opted to developed four malicious applications that perform Denial-of-Service and information theft attacks. DoS attacks against mobile phones can be categorized into the two types. The first attempts to flood and overwhelm a device by issuing fraudulent service requests, and the second attempts to drain the power resources of the device. Information theft attacks against mobile devices can be classified as targeting either transient information or static information. Transient information includes a mobile device's location, power usage patterns, and other temporal data which a mobile device typically doesn't record (Koong et al. 2008). Static information refers to any information that a mobile device intentionally and persistently maintains/sends on behalf of its owner, i.e., device identification data, contact information, phone numbers, programs and media files. Attackers may attempt to steal both types of information if they perceive it as valuable.

Tip Calculator In this study we developed the *Tip Calculator* malicious application, a calculator which unobtrusively performs a DoS attack. When a user clicks the "calculate" button to calculate the tip, the application starts a background service that waits for some time and then launches several hundreds of CPU-consuming

[3] Also known as False Alarm Rate in the intrusion detection community.

threads. The effect of this attack is an almost absolute paralysis of the device. The system becomes very unresponsive and the only effective choice is to shutdown the device (which also takes some time). An interesting observation is that the Android system often kills a CPU-consuming service but always keeps on re-launching it a few seconds later.

Schedule SMS and Lunar Lander The first malware, which belongs to the class of information-theft, includes two Android applications, *Schedule SMS* and *Lunar Lander*, exploiting the Shared-User-ID feature. In Android each application requests a set of permissions which is granted at installation time. The Shared-User-ID feature enables multiple applications to share their permission sets, provided they are all signed with the same key and explicitly request the sharing. It is noteworthy that the sharing is done behind the scenes without informing the user or asking for approval, resulting in implicit granting of permissions. The first application is Schedule SMS, a truly benign application that enables to send delayed SMS messages to people from a contact list, for which the application requests necessary permissions. The second application, Lunar Lander, is a seemingly benign game that requests no permissions. Nevertheless, once both applications are installed and the Lunar Lander obtains an ability to read the contacts and send SMS messages, it exhibits a Trojan-like behavior leaking all of contact names and phone numbers through SMS messages to a pre-defined number. This resembles RedBrowser—a Trojan masquerading as a browser that infects mobile phones running J2ME by obtaining and exploiting SMS permissions.

Snake The second information-theft application masquerades as a *Snake* game and misuses the devices camera to spy on the unsuspecting user. The Snake game requests Internet permission for uploading top scores and while depicting the game on the screen, the application is unobtrusively taking pictures and sending them to a remote server.

HTTP upload The third and last malicious information-theft application we developed, *HTTP Upload*, also steals information from the device. It exploits the fact that access to the SD-card does not require any permission. Therefore, all applications can read and write to/from the SD-card. The application requires only the Internet permission and in the background it accesses the SD card, steals its contents and sends them through the Internet to a predefined address.

In addition to malicious applications, 20 benign game and 20 benign tool applications were used, 11 of them were available on the Android framework, while the rest were obtained from the Android Market. 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.[4] In order to apply static analysis on.dex files we developed and used a.dex file parser. This parser analyzes the contents of a.dex file to extract meaningful information thereby can facilitating manual inspection and verification of Android applications (Shabtai et al.

[4]Android uses a proprietary format for Java bytecode called.dex (Dalvik Executable), designed to be more compact and memory-efficient than regular Java class files.

2009a). Examples of such information are lists of Android framework's methods, classes and types used by the application.

The aforementioned applications (i.e., 20 benign games, 20 benign tools and four malicious applications) were installed on two Android devices. The two devices were similar in the platform (HTC G1) with the same firmware and software versions. The two devices were used regularly by two different users and thus varied in the amount and type of applications installed as well as usage patterns. It was important to cross-validate the experiments among the devices in order to see if one can create a generic detection model on one device and later replicate that model to other devices. This indicates whether the selected features are dependent on the specific device and user. The malware detection 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.

Both Android devices had one user each who used all 44 applications for 10 min, while in the background the malware detection system collected new feature vectors every 2 s. 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', 'tool' or 'malicious'. The Table 7 in Appendix 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 Table 8 in Appendix). We selected the features/group of features based on their availability (i.e., the ability and ease of their extraction) and based on our subjective estimation of the features that will be most helpful in detecting a malware. Table 2 depicts the features expected to be affected by each of the four malicious applications. However, similar to any other Machine Learning application, we aimed at extracting as much features as possible in order to allow the Machine Learning algorithms to select the most informative and valuable features for the classification task. Therefore, the collected features spans over a large variety of aspects (system performance, user activity etc.)

Table 2 Features expected to be affected per application (based on the features listed in Table 8 in Appendix)

Application	Expected features to be affected
Tip Calculator	CPU Load, Operating System (e.g., context switches), Memory (e.g., garbage collection), Power (i.e., battery)
Snake	Touch screen, Keyboard, Hardware (e.g, camera), Network, Power, Operating System Memory
HTTP Upload	Network, Operating System, Memory
Schedule SMS and Lunar Lander	Application, Messaging, Touch screen, Keyboard, Operating System

Table 3 Sub-experiments description

Experiment	No. of devices	No. of detection algorithms	No. of feature selection methods	No. of top features groups	No. of datasets	No. of iterations	Total number of runs	Training/ testing performed on the same device	Testing on applications not in training set
I	2	6	3	3	10	10	10,800	+	−
II	2	6	3	3	10	4	4,320	+	+
III	2	6	3	3	10	1	1,080	−	−
IV	2	6	3	3	10	4	4,320	−	+

5.3 Agenda of experiments

The purpose of the experiments was to evaluate the ability of the proposed detection methods to distinguish between benign and malicious applications. The following set of experiment contains four segments (sub-experiments), examining the performance of the malware detection system in different situations. For each sub-experiment we used 10 datasets extracted from two different devices, on which we evaluated six detection algorithms, three feature selection methods, and three sizes of top features groups (10, 20 and 50) as presented in Table 3. As presented in Section 5.2, the datasets were collected by activating 44 applications for 10 min each. The applications and the amount of feature vectors are presented in Table 7 in Appendix.

Each one of the 10 datasets contained feature vectors of four benign applications and four malicious applications. To create the first five datasets, we divided randomly the 20 tools to five groups of size 4, while none of the tools overlapped across the different groups. For the other five datasets, we divided the 20 games to five groups of size 4 in a similar fashion (see Fig. 3). All feature vectors in a group were added to the corresponding dataset. The reason for choosing only four games or four tools is to guarantee that the different classes are equally represented in each dataset. The feature vectors for the malicious applications were collected and added to each one of the 10 datasets.

5.3.1 Experiment I

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 benign and malicious applications when training set includes all benign/malicious applications and training/testing are performed on the same device.*

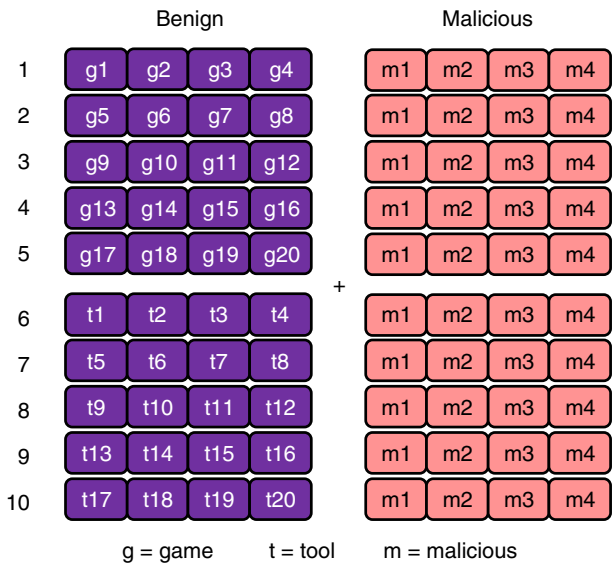

Fig. 3 Illustration of the creation of the 10 datasets

The training set contained 80% of the feature vectors of both the malicious and benign applications. The testing set contained the rest 20% feature vectors of the same malicious and benign applications (Fig. 4a). The feature vectors were assigned in a random fashion.

For each one of the 10 datasets (generated by selecting five times four different benign applications, i.e., tools and games), we iterated 10 times over each of the two devices used in the experiment, with different assignments of the training and testing set, for each configuration (see Table 3).

5.3.2 Experiment II

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 benign and malicious applications not included in the training set, when training and testing are performed on the same device*. Thus this experiment attempts to answer the first research question (Section 5.1) and understand whether it is possible to detect unknown instances of malicious applications on Android.

The training set contained feature vectors of three malicious and three benign applications. The testing set contained feature vectors of malicious and benign applications that were not included in the training set on the same device (Fig. 4b). This examined the ability of the different classifiers to detect unknown applications.

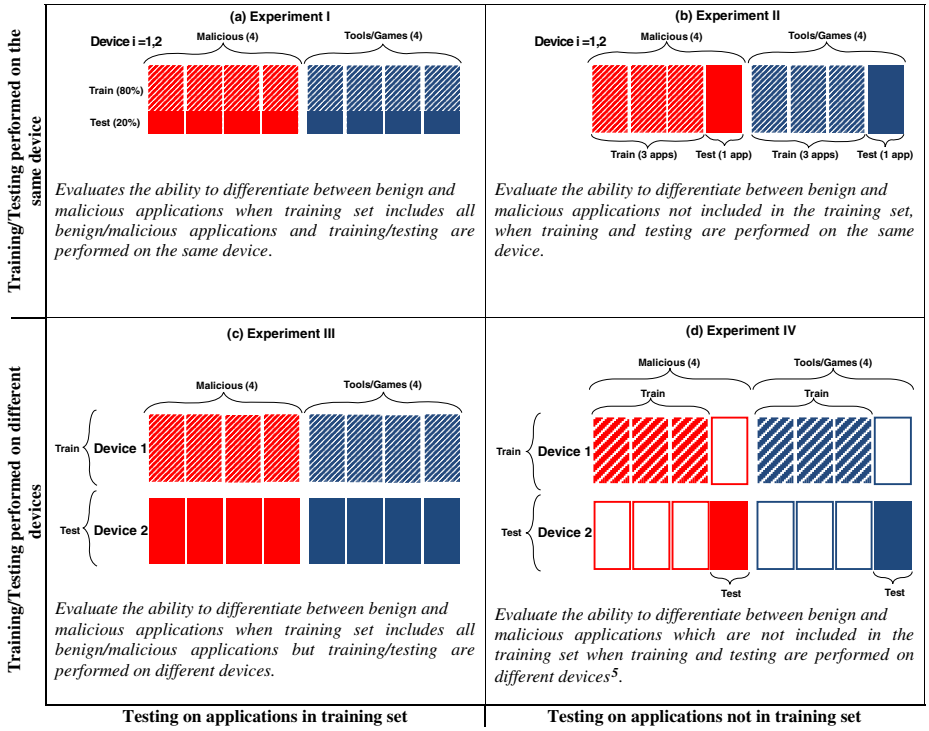

Fig. 4 Illustration of one dataset set in each experimental configuration

For each one of the 10 datasets, we iterated over each device four times, and each time a different malicious application and different benign application were not included in the training set but were included in the testing set (see Table 3).

5.3.3 Experiment III

The purpose of this experiment is to compare the ability of each combination of detection algorithm, feature selection method, and number of top features to *differentiate between benign and malicious applications when training set includes all benign/malicious applications but training/testing are performed on different devices*. Thus, this experiment attempts to answer the second research question (Section 5.1) and understand whether it is possible to detect malicious behavior on "unlearned" devices.

We divided the data to training set and testing set as follows: The training set contained all the feature vectors of the four benign applications and four malicious applications collected from the first device. The testing set contained all the feature vectors of the same four benign and malicious applications as in the training set collected from the second device which was not used for training (Fig. 4c).

We ran this experiment for each one of the 10 datasets and for each configuration. Each time the feature vectors of a different device were included in the training set and the feature vectors of the other device were used to produce the testing set (see Table 3).

5.3.4 Experiment IV

The purpose of this experiment is to compare the ability of each combination of detection algorithm, feature selection method, and number of top features to *differentiate between benign and malicious applications which are not included in the training set when training and testing are performed on different devices*. Thus, this experiment attempts to answer both the first and second research questions (Section 5.1) and understand whether it is possible to detect unknown malicious applications on "unlearned" devices.

The datasets were divided as follows: The training set contains feature vectors of three benign applications and three malicious applications collected from one of the devices. The testing set contained feature vectors collected from the second device for the remaining game/tool and malicious applications that were not included in the training set of the first device (Fig. 4d).

For each one of the 10 datasets, we iterated over each device four times, and each time different malicious application and different benign application were not included in the training set but were included in the testing set of the other device, for each configuration (see Table 3).

5.4 Experimental results

First, we wanted to answer the 3rd research question (Section 5.1) and identify the most accurate classifier for detecting malware on Android. Table 4 presents, for each experiment, the average FPR, TPR, AUC and Accuracy of the two devices when combined together, for each detector and for each benign type (tool or game).

Table 4 Average FPR, TPR, AUC and accuracy for each one of the detectors

Experiment	Detectors	Games				Tools			
		FPR	TPR	AUC	Accuracy	FPR	TPR	AUC	Accuracy
I	BN	0.000	0.985	1.000	0.993	0.000	0.983	1.000	0.992
	DTJ48	**0.000**	**0.999**	**1.000**	**1.000**	**0.001**	**0.999**	**0.999**	**0.999**
	Histogram	0.081	0.958	0.984	0.934	0.082	0.940	0.982	0.928
	Kmeans	0.115	0.697	0.791	0.813	0.175	0.681	0.753	0.774
	Logistic regression	0.001	0.999	1.000	0.999	0.001	0.998	1.000	0.999
	NB	0.007	0.926	0.995	0.962	0.009	0.901	0.991	0.947
II	BN	0.057	0.436	0.912	0.609	0.044	0.458	0.911	0.658
	DTJ48	**0.113**	**0.907**	**0.895**	**0.908**	**0.122**	**0.796**	**0.837**	**0.860**
	Histogram	0.322	0.630	0.748	0.691	0.279	0.606	0.768	0.709
	Kmeans	0.191	0.549	0.679	0.744	0.166	0.483	0.658	0.710
	Logistic regression	0.118	0.816	0.920	0.874	0.115	0.622	0.789	0.794
	NB	0.103	0.837	0.925	0.891	0.169	0.777	0.838	0.819
III	BN	0.063	0.392	0.874	0.579	0.120	0.464	0.806	0.560
	DTJ48	0.208	0.360	0.572	0.623	0.318	0.509	0.596	0.638
	Histogram	0.367	0.708	0.723	0.660	0.439	0.700	0.719	0.610
	Kmeans	0.154	0.457	0.652	0.692	0.216	0.411	0.597	0.600
	Logistic regression	0.227	0.798	0.916	0.809	0.179	0.451	0.720	0.673
	NB	**0.147**	**0.913**	**0.918**	**0.880**	**0.231**	**0.878**	**0.877**	**0.828**
IV	BN	0.056	0.171	0.735	0.441	0.121	0.216	0.678	0.403
	DTJ48	0.166	0.312	0.573	0.643	0.312	0.343	0.515	0.581
	Histogram	0.425	0.550	0.644	0.527	0.415	0.561	0.682	0.532
	Kmeans	0.167	0.374	0.603	0.669	0.202	0.374	0.586	0.615
	Logistic regression	0.210	0.608	0.803	0.759	0.257	0.311	0.563	0.595
	NB	**0.178**	**0.825**	**0.898**	**0.857**	**0.297**	**0.747**	**0.760**	**0.761**

The results show that the best detectors (classifiers) in experiments I and II are DT, Logistic Regression and NB, while in experiment III and IV the NB outperformed other classifiers.

Figure 5 depicts for each experiment, the average FPR and Accuracy of the two devices when combined together, for each detector and for each benign type (tool or game). Figure 5 indicates that when using games as the benign application type, the average detection rates, for most of the detectors, were better than when using tools (a higher accuracy and lower FPR). Overall, the behavior of detectors, however, was similar when using games or tools. For example, in the first experiment, k-Means had a lower accuracy and higher FPR relative to other detectors—for both game and tool applications. Finally, we observed that in experiment I devices 1 and 2 had similar results for BN, DT, Logistic Regression and NB. For experiments II, III and IV devices 1 and 2 had similar results for DT and NB.

Next, we wanted to answer the 4th research question (Section 5.1) and identify the best feature selection method and the top number of features. Table 5 depicts the averaged Accuracy and FPR of the three feature selections and the three top

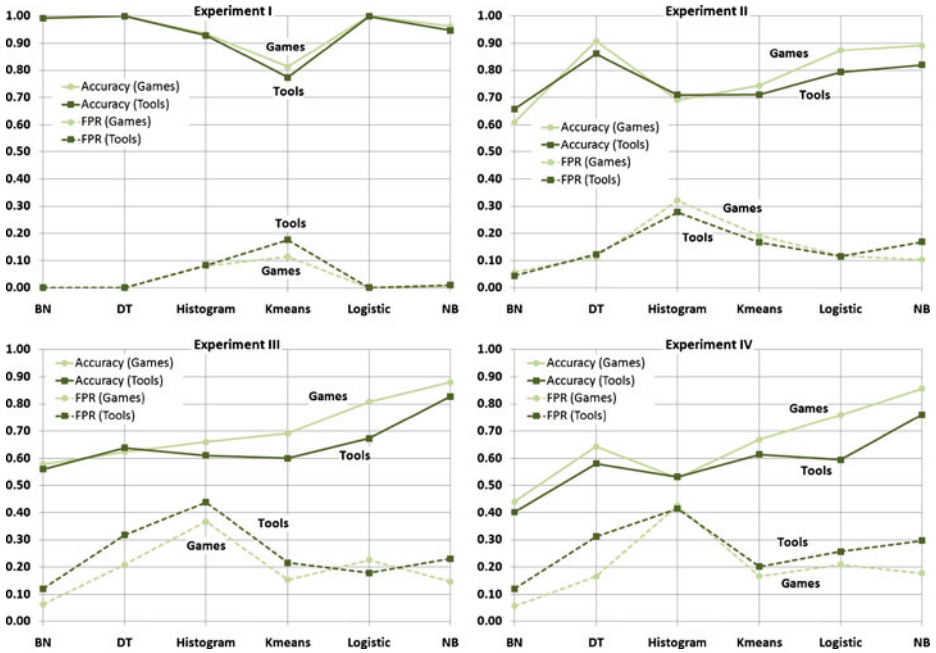

Fig. 5 Average accuracy and FPR for each one of the detectors

numbers of selected features. The result indicate that for experiment I, InfoGain with top 10 features outperformed all the others combinations. For experiments II, III and IV Fisher score with top 10 outperformed all the others. Furthermore, almost all the feature selection methods performed better when the benign type was a game. Finally, the majority of feature selection methods preformed better when top 10 features were selected than when top 20 or 50 features were selected.

Finally, Fig. 6 refers to the 5th research question (Section 5.1) and presents 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 malicious and benign applications. 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. Additionally, we omitted features with a low score and features that were chosen by only 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 Fig. 6 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: Anonymous_Pages, Garbage_Collections, Total_Entities, Battery_Temp, Running_Processes and Mapped_Pages. These features, excluding Total_Entities and Mapped_Pages, were high-ranked by Fisher Score as well.

Table 5 Averaged Accuracy and FPR for each one of the feature selection methods and top features

Experiment	Feature selection method	Games						Tools					
		FPR			Accuracy			FPR			Accuracy		
		10	20	50	10	20	50	10	20	50	10	20	50
I	ChiSquare	0.023	0.034	0.047	0.963	0.945	0.935	0.025	0.035	0.059	0.951	0.941	0.930
	FisherScore	0.021	0.027	0.052	0.962	0.948	0.933	0.052	0.050	0.063	0.945	0.934	0.929
	InfoGain	**0.020**	0.027	0.047	**0.968**	0.960	0.936	**0.025**	0.034	0.060	**0.952**	0.948	0.930
II	ChiSquare	0.140	0.136	0.188	0.769	0.779	0.771	0.135	0.162	0.148	0.765	0.728	0.754
	FisherScore	**0.119**	0.119	0.181	**0.851**	0.818	0.774	**0.136**	0.164	0.150	**0.794**	0.783	0.763
	InfoGain	0.149	0.137	0.188	0.763	0.777	0.771	0.135	0.163	0.148	0.765	0.721	0.752
III	ChiSquare	0.168	0.209	0.213	0.722	0.662	0.702	0.295	0.248	0.229	0.616	0.613	0.666
	FisherScore	**0.173**	0.164	0.193	**0.799**	0.754	0.715	**0.208**	0.222	0.246	**0.721**	0.718	0.681
	InfoGain	0.196	0.209	0.203	0.640	0.662	0.708	0.321	0.249	0.235	0.574	0.615	0.662
IV	ChiSquare	0.193	0.196	0.243	0.662	0.617	0.619	0.292	0.244	0.274	0.573	0.554	0.558
	FisherScore	**0.176**	0.172	0.207	**0.764**	0.681	0.644	**0.242**	0.241	0.287	**0.690**	0.636	0.577
	InfoGain	0.169	0.196	0.240	0.622	0.617	0.619	0.296	0.256	0.274	0.531	0.552	0.557

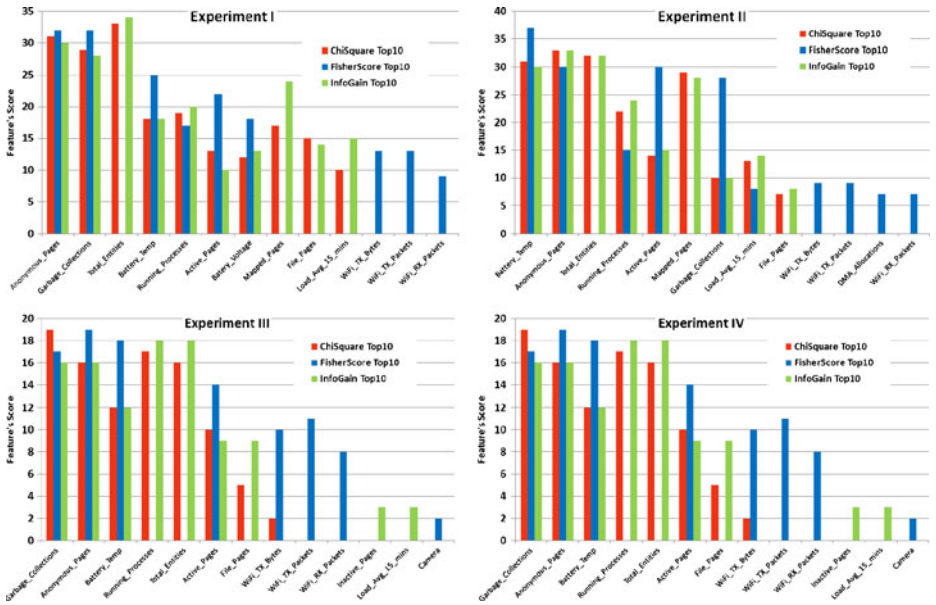

Fig. 6 Best selected features

5.5 Performance overhead

Intrusion detection systems (IDSs) must maximize the realization of security goals while minimizing costs. Lee et al. (2002) examined the major cost factors associated with an IDS which include development cost, operational cost, damage cost due to successful intrusions, and the cost of manual and automated response to intrusions. Their experiments indicate that cost-sensitive modeling and deployment techniques are effective in reducing the overall cost of intrusion detection. Lee and Xiang (2001) propose to use several information-theoretic measures, namely, entropy, conditional entropy, relative conditional entropy, information gain, and information cost for anomaly detection.

In our study, order to check the impact of the malware detector on the device resources, we measured the memory and CPU consumption as well as battery exhaustion. We maintained the memory consumption below an acceptable threshold by ensuring that no information is kept in memory and only last three samples are stored for visualization purposes. The memory consumption of the application was steady in the interval 16,780 Kb $\pm$ 32 (which is approximately 8.5% of the device's RAM). The CPU consumption was in the interval 5.52% $\pm$ 2.11 and peaks occurred during the extraction and analysis of features. We also measured the battery exhaustion with and without the Andromaly application. In this test, when running the malware detector, we measured 30 features and applied eight different classifiers. The sampling interval was set to 5 s and the eight classifiers were launched sequentially (i.e., one-by-one) during each 5-s interval. This test exhibits an approximate 10% performance degradation when running the malware detector. However, as the evaluation results indicate, detection can be performed using only

10 features and a smaller number of classifiers thus the results of this test depict the impact on the battery in the worst case scenario.

6 Discussion

Table 6 presents the best configurations, which outperformed all other configurations in terms of ability to differentiate between malicious and benign applications for each one of the four sub-experiments (averaged over the two devices, both benign application types and all iterations).

Figure 7 depicts the ROC graphs of an arbitrary run on device #1 for experiments III and IV. These graphs were plotted for the best averaged configuration; i.e., NB trained on the top 10 features selected by the Fisher Score algorithm.

Interestingly, in order to provide the best detection performance, sub-experiments II–IV employed a similar configuration, with NB as the detector and Fisher Score—top 10 for feature selection. NB applies a simple linear model that simplifies the classification process by assuming that features are conditionally independent. This fact along with the observation that in each of the experiments only top 10 features were required to achieve the best performance demonstrates the practical feasibility of the proposed framework to Android devices.

When observing all sub-experiments we can conclude that the NB and Logistic Regression were superior over other classifiers in the majority of the configurations. Additionally, Fisher Score with 10 top features selected is the best setting in most of the cases.

Another interesting fact is that in all experiments it was easier to distinguish between malicious and benign applications when the benign application was a game, as opposed to a tool application. We hypothesize that it is due to the fact that games have more unique features compared to other types of benign applications, and also have a high degree of similarity compared to other application groups.

Additionally, there is high similarity in the features that were selected on each one of the sub experiments. We suggest that it is due to a unique and persistent behavior of malicious applications across devices. The most common highly ranked features across all the sub-experiments were Anonymous_Pages, Garbage_Collections, Battery_Temp, Total_Entities, Active_Pages and Running_Processes.

Table 6 Averaged FPR, TPR, Accuracy and AUC of the best configuration in experiments I–IV

Experiment	Benign type	Best configuration	TRP	FPR	AUC	Accuracy
1	Games	DT\J48 InfoGain 10	0.999	0.000	0.999	0.999
1	Tools	DT\J48 InfoGain 10	0.999	0.001	0.999	0.999
1	Average	DT\J48 InfoGain 10	0.999	0.001	0.999	0.999
2	Games	NB FisherScore 10	0.842	0.065	0.948	0.931
2	Tools	NB FisherScore 10	0.790	0.108	0.914	0.869
2	Average	NB FisherScore 10	0.847	0.123	0.913	0.873
3	Games	NB FisherScore 10	0.919	0.148	0.932	0.882
3	Tools	NB FisherScore 10	0.834	0.060	0.935	0.887
3	Average	NB FisherScore 10	0.876	0.104	0.933	0.885
4	Games	NB FisherScore 10	0.853	0.162	0.930	0.880
4	Tools	NB FisherScore 10	0.734	0.090	0.831	0.868
4	Average	NB FisherScore 10	0.794	0.126	0.881	0.874

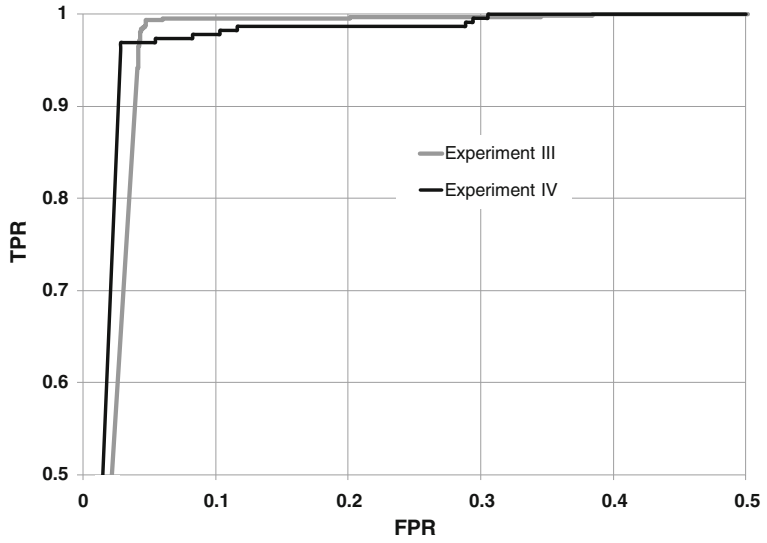

Fig. 7 ROC graphs for experiments III and IV

Looking at the performance of classifiers on each of the two devices separately, it is evident that they exhibit similar performance. This supports the external validity of our experiments, indicating that the selected features in our experiments are not user-, or configuration-dependent, that is, we can train the classifiers on a set of devices and can still detect effectively on other devices.

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 (Schmidt et al. 2009). This observation strengthens the applicability of anomaly detection techniques for malware detection on mobile devices in general.

Andromaly differs from earlier efforts enhance Android security such as *Kirin* system (Enck et al. 2008, 2009) and Secure Application INTeraction (SAINT) (Hwang et al. 2009). These two systems presented an installer and security framework that realize an overlay layer on top of Android's standard application permission mechanism. This layer allows applications to exert fine-grained control over the assignment of permissions through explicit policies.

This research, however, suffers from two limitations. First and foremost, Android is a relatively new platform and at the time we had performed our experiments no malware were known; therefore we opted to test artificially-created malware in lieu of real malware. In our opinion it should not stop researchers from exploring new methods in order to maintain pace with emerging malware. Symbian-based devices are an unequivocal example for why it is important that researches will precede malware appearance. As a case in point, Cabir, which was notable for being the first genuine worm threat to hit mobile phones, targeted Symbian Series 60-compatible devices only in June 2004, only 3 years after the first Edition of Symbian Series 60 smartphone platforms was first introduced (Shih et al. 2008). Therefore, it would be interesting later on, to test the proposed method on new, "real" instances of Android malware. Our comprehensive security assessment of

the Android framework indicates that malware penetration to the device is likely to happen, not only at the Linux level but in the application level (Java) while exploiting the Android permission mechanism and thus exploring methods for protecting the Android framework is essential (Shabtai et al. 2009d, 2010b).

We tried to overcome this limitation as much as possible by: 1) using separate research groups one for developing and evaluating the malware detector and the other for developing malware for Android; 2) development of malware for Android was done after a thorough 18-months research of Android platform, its vulnerabilities, and weakness as well as malware from other mobile platforms; 3) moreover, we used a large and heterogeneous pool of features (88) in order to avoid any bias. Most of the features were low level system features independent of specific applications.

A second limitation stems from the fact that the variety of malware applications was quite small (using only four malware instances). This is again due to the fact that Android is a new platform and no known malware yet exists. This limitation is somewhat mitigated by the fact that we performed a preliminary evaluation of the proposed detection model on 20 tools and 20 games across two Android devices. Similar four sub-experiments were conducted and yielded similar performance, thereby providing a positive indication about the validity of our findings and about the ability of such methods to learn and model the behavior of an Android application and potentially detect malicious applications, even when the number of sampled features is small. One important and interesting note is that the four malware instances used in the evaluation were significantly different in behavior and in impact/effect on the system and yet yielded relatively good results.

7 Conclusions and future work

Since Android has been introduced, it has been (and still is) explored for its inherent security mechanisms, and several targeted security solutions where proposed to augment these mechanisms. From our assessment of Android's security (Shabtai et al. 2009c, 2010a), we believe that additional security mechanisms should be applied to Android. Furthermore, similar to the PC platform, there is no "silver-bullet" when dealing with security. A security suite for mobile devices or smartphones (especially open-source) such as Android include a collection of tools operating in collaboration. This includes: signature-based anti-virus, firewalling capabilities, better access-control mechanism and also a malware/intrusion detection platform.

In this paper we presented a malware detection 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, derived locally on each device, to such a centralized location. This can be useful in detection of malware propagation patterns across a community of mobile devices. As stated by Rich Cannings,[5] Google's Android Security Leader, the Android Market place was chosen to be the place for reporting security issues by users. Users can mark applications as harmful, thereby triggering a security team to launch an investigation

[5] http://www.usenix.org/events/sec09/tech/tech.html#cannings

(as in the case of the MemoryUp application[6]). Andromaly can be used for reporting suspicious behavior of applications to the Android Market.

Andromaly is open and modular and, as such, can easily accommodate multiple malware detection techniques (e.g., rule-based inferences (Moreau et al. 1997), and the behavioral checker approach (Samfat and Molva 1997)). Overall, our empirical findings indicate that using Machine Learning algorithms for anomaly detection is a viable approach for enhancing security of Android devices. We also found that Feature Extractors are in the critical execution loop and as such must be optimized aggressively. Some of the mechanisms used to prevent them from becoming performance bottle necks are: using a small number of features for the detection; implementing resource exhaustive feature extractors as native code; and sending only the required features to each processor to reduce the Binder communication overhead. Furthermore, the fact that the detection can still be effective even when using a small number features and simple detection algorithms ensures that stringent resource constraints (i.e., CPU, battery) on the device are met.

The proposed detection approach is recommended for detecting continuous attacks (e.g., DoS, worm infection) and needs to be trained on a broad range of examples. Since one of the primary requirements of such detector is to maintain a low false alarm rate, alerting a user on each instance of detected anomaly is not acceptable; therefore the prevalent approach would be to dispatch an alert only in case the anomaly persists. This approach however, will fail to detect instantaneous and abrupt attacks. Thus, the main conclusion emanating from this study is that a more suitable approach when implementing a malware detection system for Android would be to mesh the following two methods: classification and anomaly detection and misuse-based detectors (e.g., rule-based, knowledge-based). When they operate in a synergy they can yield an optimal recall rate and low false alarms.

We identified several challenges that complicate the use of anomaly detection in an Android platform: (1) malicious activities can be very short; thus not providing sufficient data to learn from or detect (e.g., the SMS leakage malware); (2) there are not enough malicious applications to use in the training phase; and (3) the malicious behavior may vary between attacks, resulting in many types (i.e., classes) of malicious behaviors (4) The small number of malicious applications introduced us with the class imbalance problem. The class imbalance problem occurs when one of the classes is represented by a very small number of cases compared to the other classes. We decided to cope with this problem by under-sampling the benign class, in other words, using only part of the benign applications that were generated for the first set of experiments, and duplicating the vectors of malicious applications.

Several avenues can be explored for future research. First and foremost we can alert about a detected anomaly when it persists for a long time. Mechanisms such as a Leaky Bucket, Moving Average, and the Knowledge-Based Temporal Abstraction (KBTA) (Shabtai et al. 2009b, 2010a) can assist in this task. Second, we can detect features with high variance between different devices and remove them, and also add new and valuable features such as system calls. 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 10 min rather than the level of the battery at a certain point in time), or logging sequences of events (e.g., a Boolean feature

[6] http://www.geek.com/articles/mobile/google-says-that-memoryup-has-no-malware-20090128/

that is true if there was an access to an SD-card concurrently with a high volume of network traffic via Wi-Fi). Fourth, we would like to test the proposed framework on different/newer handset firmware in order to understand the impact on the performance. Fifth, 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. Finally, unsupervised classifiers can be used instead of the supervised classifiers used in this study.

Appendix

Table 7 List of used applications

Class	Application name	No. of vectors (device 1)	No. of vectors (device 2)
Malicious	Lunar Lander (SMS abuse)	206	313
	Snake (camera abuse)	227	240
	HTTP uploader (SD-card leakage)	175	214
	Tip Calculator (DoS)	196	210
	(Total)	804 (6%)	977 (9%)
Benign (tools)	Browser	307	274
	Calculator	296	251
	Calendar	319	233
	Camera	302	251
	Contacts	303	230
	eMail	219	250
	Messaging	371	245
	File Manager	295	235
	Maps	342	216
	SMS	322	247
	Playing Music	304	251
	My Tracks	356	233
	Note Everything	329	212
	Oxford Dictionary	323	275
	Pdf Viewer	280	249
	Phonalyzr	304	240
	Phone	125	224
	Tasks	300	226
	Voice Memo	312	230
	Weather File Manager	372	242
	(Total)	6,081 (44%)	4,814 (44%)
Benign (games)	Armageddonoid	343	222
	Battle for Mars	544	208
	Bonsai	355	287
	Bread Factory	300	208
	Connect4	308	204
	Flying High	279	250
	Froggo	308	185
	Hang'em High	342	348
	Lexic	354	266
	Mine Sweeper	342	267
	Pairup	410	275
	PicaCross Express	300	357
	SmartTacToe	298	233
	Snake	294	254
	Solitaire	334	378
	Switcher	303	231
	TankAce	336	287
	ThrottleCopter	321	230
	Trap	454	245
	WordPops	301	314
	(Total)	6,826 (50%)	5,249 (48%)

Table 8 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		

References

Adam, P. F., Chaudhuri, A., & Foster, J. S. (2009). SCanDroid: Automated security certification of android applications. In *IEEE symposium of security and privacy*.

Bose, A., Hu, X., Shin, K. G., & Park, T. (2008). Behavioral detection of malware on mobile handsets. In *Proc. of the 6th international conference on mobile systems, applications, and services*.

Botha, R. A., Furnell, S. M., & Clarke, N. L. (2009). From desktop to mobile: Examining the security experience. *Computer & Security, 28*, 130–137.

Buennemeyer, T. K., et al. (2008). Mobile device profiling and intrusion detection using smart batteries. In *International conference on system sciences* (pp. 296–296).

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys, 41*(3), 1–58.

Chaudhuri, A. (2009). Language-based security on android. In *ACM workshop on programming languages and analysis for security (PLAS)* (pp. 1–7).

Cheng, J., Wong, S. H., Yang, H., & Lu, S. (2007). SmartSiren: Virus detection and alert for smartphones. In *Proceedings of the 5th international conference on mobile systems, applications and services*.

Dagon, C., Martin, T., & Starner, T. (2004). Mobile phones as computing devices the viruses are coming. *Pervasive Computing, 3*, 11–15.

Domingos, P., & Pazzani, M. (1997). On the optimality of simple Bayesian classifier under zero-one loss. *Machine Learning, 29*, 103–130.

Egele, M., Krugel, C., Kirda, E., Yin, H., & Song, D. (2007). Dynamic spyware analysis. In *USENIX annual technical conference* (pp. 233–246).

Emm, D. (2006). Mobile malware – new avenues. *Network Security, 2006*(11), 4–6.

Enck, W., Ongtang, M., & McDaniel, P. (2008). *Mitigating android software misuse before it happens.* Tech. report NAS-TR-0094–2008, Network and Security Research Ctr., Dept. Computer Science and Eng., Pennsylvania State Univ.

Enck, W., Ongtang, M., & McDaniel, P. (2009). Understanding android security. *IEEE Security & Privacy Magazine, 7*(1), 50–57.

Endler, D. (1998). Intrusion detection: Applying machine learning to solaris audit data. In *Proceedings of the 14th annual computer security applications conference*.

Garcia-Teodoro, P., Diaz-Verdejo, J., Macia-Fernandez, G., & Vazquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security, 28*(1–2), 18–28.

Golub, T., et al. (1999). Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science, 286*, 531–537.

Griffin, K., Schneider, S., Hu, X., & Chiueh, T. (2009). Automatic generation of string signatures for malware detection. In *Proc. of the 12th international symposium on recent advances in intrusion detection*.

Gryaznov, D. (1999). Scanners of the year 2000: Heuritics. *The 5th international virus bulletin*.

Guo, C., Wang, H. J., & Zhu, W. (2004). Smart-phone attacks and defenses. In *HotNets III*.

Hwang, S. S., Cho, S., & Park, S. (2009). Keystroke dynamics-based authentication for mobile devices. *Computer & Security, 28*, 85–93.

Imam, I. F., Michalski, R. S., & Kerschberg, L. (1993). Discovering attribute dependence in databases by integrating symbolic learning and statistical analysis techniques. In *Proceeding of the AAAI-93 workshop on knowledge discovery in databases*.

Jacob, G., Debar, H., & Filiol, E. (2008). Behavioral detection of malware: From a survey towards an established taxonomy. *Journal in Computer Virology, 4*, 251–266.

Jacoby, G. A., & Davis, N. J. (2004). Battery-based intrusion detection. In *Global telecommunications conference (GLOBECOM'04)*.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering. *ACM Computing Surveys, 31*(3):264–296.

John, G. H., & Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In *Proc. of the conference on uncertainty in artificial intelligence* (pp. 338–345).

Kim, H., Smith, J., & Shin, K. G. (2008). Detecting energy-greedy anomalies and mobile malware variants. In *Proceeding of the 6th international conference on mobile systems, applications, and services*.

Koong, K. S., Liu, L. C., Bai, S., & Lin, B. (2008). Identity theft in the USA: Evidence from 2002 to 2006. *International Journal of Mobile Communications, 6*(2), 199–216.

Leavitt, N. (2005). Mobile phones: The next frontier for hackers? *Computer, 38*(4), 20–23.

Lee, W., & Xiang, D. (2001). Information-theoretic measures for anomaly detection. In *Proc. of the IEEE symposium on security and privacy* (pp. 130–143).

Lee, W., Stolfo, S., & Mok, K. (1999). A data mining framework for building intrusion detection models. In *Proc. of the 1999 IEEE symposium on security and privacy*. Oakland.

Lee, W., Fan, W., Miller, M., Stolfo, S., & Zadok, E. (2002). Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security, 10*(1–2), 5–22.

Menahem, E., Shabtai, A., Rokach, L., & Elovici, Y. (2008). Improving malware detection by applying multi-inducer ensemble. *Computational Statistics and Data Analysis, 53*(4), 1483–1494.

Miettinen, M., Halonen, P., & Hätönen, K. (2006). Host-based intrusion detection for advanced mobile devices. In *Proc. of the 20th international conference on advanced information networking and applications*.

Mitchell, T. (1997). *Machine learning*. New York: McGraw-Hill.

Moreau, Y., Preneel, B., Burge, P., Shawe-Taylor, J., Stoermann, C., & Cooke, C. (1997). Novel techniques for fraud detection in mobile telecommunication networks. In *ACTS mobile summit*.

Moser, A., Kruegel, C., & Kirda, E. (2007). Limits of static analysis for malware detection. In *Annual computer security applications conference* (pp. 421–430).

Moskovitch, R., Elovici, Y., & Rokach, L. (2008). Detection of unknown computer worms based on behavioral classification of the host. *Computational Statistics and Data Analysis, 52*(9), 4544–4566.

Muthukumaran, D., et al. (2008). Measuring integrity on mobile phone systems. In *Proceedings of the 13th ACM symposium on access control models and technologies*.

Nash, D. C., et al. (2005). Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices. In *Pervasive computing and communications workshops*.

Neter, J., Kutner, M. H., Nachtsheim, C. J., & Wasserman, W. (1996). *Applied linear statistical models*. McGraw-Hill.

Ongtang, M., McLaughlin, S., Enck, W., & McDaniel, P. (2009). Semantically rich application-centric security in android. In *Proceedings of the 25th annual computer security applications conference (ACSAC)*. Honolulu.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Massachusetts: Morgan Kaufmann.

Piercy, M. (2004). Embedded devices next on the virus target list. *IEEE Electronics Systems and Software, 2*, 42–43.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Francisco: Morgan Kaufmann.

Rieck, K., Holz, T., Willems, C., Düssel, P., & Laskov, P. (2008). Learning and classification of malware behavior. In *Proc. of the conference on detection of intrusions and malware & vulnerability assessment* (pp. 108–125).

Russel, S., & Norvig, P. (2002). *Artificial intelligence: A modern approach.* Prentice Hall.

Samfat, D., & Molva, R. (1997). IDAMN: An intrusion detection architecture for mobile networks. *IEEE Journal on Selected Areas in Communications, 15*(7), 1373–1380.

Schmidt, A. D., Schmidt, H. G., Yüksel, K. A., Kiraz, O., Camptepe, S. A., & Albayrak, S. (2008). Enhancing security of linux-based android devices. In *Proc. of the 15th international linux system technology conference*.

Schmidt, A. D., Peters, F., Lamour, F., Scheel, C., Camtepe, S. A., & Albayrak, S. (2009). Monitoring smartphones for anomaly detection. *Mobile Networks and Applications (MONET), 14*(1), 92–106.

Shabtai, A., Fledel, Y., & Elovici, Y. (2009a). Detecting malicious applications on android by applying machine learning classifiers to static features (Poster). *Presented in the 25th annual computer security applications conference (ACSAC)*. Honolulu, Hawaii.

Shabtai, A., Fledel, Y., Elovici, Y., & Shahar, Y. (2009b). Knowledge-based temporal abstraction in clinical domains. *Journal in Computer Virology, 8*(3), 267–298.

Shabtai, A., Moskovitch, R., Elovici, Y., & Glezer, C. (2009c). Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *Information Security Technical Report, 14*(1):1–34.

Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., & Dolev, S. (2009d). *Google android: A state-of-the-art review of security mechanisms*. CoRR abs/0912.5101.

Shabtai, A., Kanonov, U., & Elovici, Y. (2010a). Intrusion detection on mobile devices using the knowledge based temporal-abstraction method. *Journal of Systems and Software, 83*(8), 1524–1537.

Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S., & Glezer, C. (2010b) Google android: A comprehensive security assessment. *IEEE Security and Privacy Magazine*. doi:10.1109/MSP.2010.2.

Shannon, C. E. (1948). The mathematical theory of communication. *The Bell system Technical Journal, 27*(3), 379–423.

Shih, D. H., Lin, B., Chiang, H. S., & Shih, M. H. (2008). Security aspects of mobile phone virus: A critical survey. *Industrial Management & Data Systems, 108*(4), 478–494.

Yap, T. S., & Ewe, H. T. (2005). A mobile phone malicious software detection model with behavior checker. *Lecture Notes in Computer Science, 3597*, 57–65.

Yin, H., Song, D., Egele, M., Krugel, C., & Kirda, E. (2007). Panorama: Capturing system-wide information flow for malware detection and analysis. In *ACM conference on computer and communications security*.