Jang *et al. SpringerPlus (2016) 5:273*
DOI 10.1186/s40064-016-1861-x

SpringerPlus
a SpringerOpen Journal

RESEARCH Open Access

Detecting and classifying method based on similarity matching of Android malware behavior with profile

Jae-wook Jang[1], Jaesung Yun[1], Aziz Mohaisen[2], Jiyoung Woo[1] and Huy Kang Kim[1*]

*Correspondence:
cenda@korea.ac.kr
[1] Graduate School of Information Security, Korea University, Seoul, Republic of Korea
Full list of author information is available at the end of the article

Abstract

Mass-market mobile security threats have increased recently due to the growth of mobile technologies and the popularity of mobile devices. Accordingly, techniques have been introduced for identifying, classifying, and defending against mobile threats utilizing static, dynamic, on-device, and off-device techniques. Static techniques are easy to evade, while dynamic techniques are expensive. On-device techniques are evasion, while off-device techniques need being always online. To address some of those shortcomings, we introduce Andro-profiler, a hybrid behavior based analysis and classification system for mobile malware. Andro-profiler main goals are efficiency, scalability, and accuracy. For that, Andro-profiler classifies malware by exploiting the behavior profiling extracted from the integrated system logs including system calls. Andro-profiler executes a malicious application on an emulator in order to generate the integrated system logs, and creates human-readable behavior profiles by analyzing the integrated system logs. By comparing the behavior profile of malicious application with representative behavior profile for each malware family using a weighted similarity matching technique, Andro-profiler detects and classifies it into malware families. The experiment results demonstrate that Andro-profiler is scalable, performs well in detecting and classifying malware with accuracy greater than 98 %, outperforms the existing state-of-the-art work, and is capable of identifying 0-day mobile malware samples.

Keywords: Behavior profiling, Similarity, System call, Android, Malware

Background

The explosive growth in the number of mobile devices running the Android platform has attracted the attention of hackers for the wealth of sensitive information that are usually stored on mobile devices, including phone numbers, short messages, confidential emails and correspondences, and banking information and credentials. The availability of this information in many mass-market mobile devices makes them a desirable target for hackers, who excelled at developing a large number of mobile malicious software (malware), making the security of mobile devices one of the most important and challenging areas of research. For example, According to a report by McAfee, the total number of mobile malware continued its linear climb as it broke 8 million in the second quarter of 2015, and increased by 17 % over the first quarter of the same year (McAfee

2015). Moreover, new malware families and variants were reported to appear approximately 1 million times in the same quarter. To address this trend, antivirus (AV) vendors analyze a large number of malware samples daily in order to prevent them from spreading widely and to guide users on disinfection and risk management by classifying malware into broad families.

Mobile as well as traditional malware analysis for detection and classification falls into two broad types: static and dynamic analysis. In static analysis, strings of bytes associated with malware samples are discovered through reverse engineering and used as a signature for identifying malicious software. Although fast and efficient, static techniques are often prone to high false negative rates due to evolution in code basis and code repacking. Furthermore, additional cost of those techniques is required for reverse engineering to generate reliable and meaningful signatures.

On the other hand, dynamic and behavior based analysis aims to provide methods for effectively and efficiently extracting unique patterns of each malware family based on its behavior. Malware samples of the same family often use the same code base, provide the same functionality using the same order of behavioral events (Mohaisen et al. 2014), and so on. In analyzing mobile malware, unique behavior patterns can be represented by various symbols (e.g., permission set, API call, and system call) and used to identify malware families. To this end, researchers previously proposed various detection and classification methods for malware analysis based on their behavior, including permission-based, API call-based and system call-based methods. Permission-based detection methods are not efficient in classifying benign applications as benign, since relevant rule sets only focus on detecting the malware. API call-based detection methods cannot generate distinct signatures until decompilation or disassembly process is completed, which is often expensive. System call-based detection methods can more accurately detect malicious behavior than other methods, since it is impossible to modify original functionality of system calls: malware creators always attempt to disguise malicious behavior as normal behavior. However, proposed methods in this category mainly deal with frequency of system calls well presented in malware. The number of invoked system calls is usually small, and most of the system calls used in malware (e.g., `read()`, `write()`) are also observed in both benign applications, affecting the accuracy of those methods. To this end, one needs to consider more features, such as arguments in the system call and network activities, to enhance malware detection and classification via behavior profiling.

To overcome the drawbacks in previous methods, we propose a feature-rich anti-malware system based on behavior profiling called Andro-profiler. Our proposed behavior profiling system comprises mobile devices and a remote server to facilitate profiling, and adopts profiling method in the malware analysis domain. We exploit system calls, including their arguments provided by Loadable Kernel Module (LKM) and system logs (e.g., SMS, call, and network I/O) provided by Droidbox (2011) as feature vectors for malware characterization. We define system calls and system logs as integrated system logs from which we directly infer behavior patterns representation using the concept of behavior profiling of Bayer et al. (2009). We assume that: (a) malware samples have unique malicious behavior patterns, (b) malicious behavior is determined by system calls, and (c) such system call set has influence on the behavior of the program (malware). We prepare

representative behavior profile for each malware family represented by integrated system logs including system calls, their arguments, and system logs of Droidbox—an analysis system we utilize in this work. We construct the behavior profile of each malware sample through its integrated system logs by executing it on an emulator. Then, by comparing the behavior profiles across samples, we can detect and classify malware samples into related families.

Contribution The main contributions of this paper are as follows:

1. We propose a novel anti-malware system based on behavior profiling called Andro-profiler. We classify malware by exploiting the behavior profiling extracted from integrated system logs. Our method captures the behavior profiling by converting integrated system logs into human-readable contexts, which helps analysts analyze malware intuitively.
2. Andro-profiler enables AV vendors to react to many species of malicious samples by classifying and matching them with those previously detected quickly and efficiently. Our system can help detect new malware including existing malware's variants and 0-day exploits. This is further highlighted through in-depth experiments using real-world malware samples.
3. Our proposed method is robust, and can be extended with additional features that depict the unique behavior patterns of malware. Our method can easily employ static analysis technique to capture malicious behavior, in combination of the dynamic behavior, which is shown to outperform existing techniques in the literature. This feature of our work is highlighted by a comparison with the prior literature, experimentally.

The rest of this paper is organized as follows. Previous work is introduced in "Related work". Our profiling method is explained in "Behavior profiling" and our proposed architecture is introduced in "Andro-profiler: an anti-malware system". The experimental results are presented in "Performance evaluation". And the limitation of our approach is discussed in "Limitations". In "Conclusion and future work", we conclude the paper and outline possible directions for future work.

Related work

Based on where the scan and monitoring of the mobile malware takes place, malware analysis methods are classified into three types: detection methods on the mobile device, detection methods outside the mobile device, and hybrid detection methods. We classify the literature based on the type of the malicious behavior into permission-based and footprint-based methods. Footprint-based methods include system call-based, API call-based, decompiled code-based, and XML information-based methods. The detection methods on mobile device scan malicious behavior patterns on the mobile device and return the analysis results to the user. However, those approaches do not consider the resource constraints on the mobile device: low computing power and limited battery life, affect their usability and user experience. The detection methods outside mobile device execute detection algorithms on an emulator or a real device running the targeted applications, and conduct static or dynamic analysis for determining the nature of

those applications. Those approaches do not need to consider resource constraints, but cannot respond to new malware families quickly. To overcome the drawbacks in both approaches, hybrid approaches have been introduced in mobile malware analysis. Client modules deployed on mobile devices collect information related to installed applications on those devices and send the information to a remote server. The remote server then analyzes log files using their detection algorithms of choice, while not impeding usability and user experience. Table 1 summarizes the various malware detection or classification methods in the literature. In the following, we elaborate on some of the related works in each category.

Detection methods on mobile devices

Previous work in this category has introduced malware detection methods that can execute applications on devices, providing online detection. Enck et al. (2009) proposed the Kirin security service, which performs lightweight certification of applications to mitigate malware at installation time. Kirin examined the requested permissions of applications, compared them with self-defined security rules, and determined whether malicious activities were carried out or not. In order to do that, they relied on permissions given in a manifest file, `Androidmanifest.xml`. However, application developers tend to excessively declare permissions in a manifest file, although the application does not actually need all of the permissions. To that end, those methods produce low accuracy. Pearce et al. (2012) introduced AdDroid, in which they separated advertising

Table 1 Various malware detection / classification methods in previous works

Approach	Method	Feature	Previous works
Detection on mobile device	Permission	Permission	Enck et al. (2009) and Pearce et al. (2012)
	Footprint	System resources	Shabtai and Elovici (2010) and Bugiel et al. (2012)
		Taint tracing	Enck et al. (2010)
		Event log, system call	Bose et al. (2008)
Detection outside mobile device	Permission	Permission	Peng et al. (2012)
	Footprint	System call, disassembled code	Blasing et al. (2010)
		System call, interaction log	Reina et al. (2013)
		System/API call, taint tracing	Rastogi et al. (2013)
	Permission + footprint	Permission, API call	Yang et al. (2012)
		Permission, API call, XML information	Grace et al. (2012), Wu et al. (2012) and Arp et al. (2014)
		Permission, API call, system call, XML information, disassembled code	Yan and Yin (2012), Zhou et al. (2012), Spreitzenbarth et al. (2013), Weichselbaum et al. (2014) and Vidas et al. (2014)
Hybrid	Footprint	System call	Burguera et al. (2011) and Isohara et al. (2011)
		Function call	Schmidt et al. (2009)
	Permission + footprint	Certificate, permission, disassembled code, XML information	Jang et al. (2015) and Kang et al. (2015)

permissions for the Android platform. In AdDroid, the host application and the core advertising code ran in isolated environment, where applications using AdDroid did not send sensitive information to advertisement server anymore. However, AdDroid did not consider information leakage unrelated to advertisement, which is the case in the majority of malware.

Shabtai and Elovici (2010) proposed Andromaly, a behavior-based detection framework for Android-based mobile devices. Andromaly is a host-based intrusion detection system that continuously monitored various resources and classified malicious applications using a machine learning algorithm. Bugiel et al. (2012) proposed Xmandroid, a system-centric and policy-driven runtime monitoring system that regulates communications between applications. Based on heuristic analysis, the authors identified attack patterns and classified malicious applications. These proposed methods, however, require a significant hardware capacity (e.g., CPU, RAM, and battery life) in order to monitor all resources comprehensively.

Enck et al. (2010) proposed Taintdroid, an extension to the Android mobile-phone platform that tracks the flow of sensitive information through third-party applications. If tainted data left the Android device, Taintdroid provided a report logging the leaked data, where the data is sent and which application leaked it. Taintdroid focused on information leakage, and then an emulator such as Droidbox embedded Taintdroid and tracked information leakage.

Bose et al. (2008) proposed a signature-based detection method for the Symbian operating system. The method is a two-stage mapping technique consisting of extraction process and representation process that constructed these signatures at run-time from the monitored system events and system calls. The method used temporal logic to detect malicious activity over time that matched a set of signatures represented as a sequence of events. However, the method needed to obtain root privileges to access the kernel, and required sufficient hardware capacity to extract system calls and convert related features into signatures.

Detection methods outside mobile devices

Previous work in this category introduced malware detection methods that execute relevant applications outside the device, providing offline detection. These methods execute their detection algorithms on an emulator or a real device other than the host device. Thus they are not constrained by constraints of real devices, and do not impede usability and user experience.

Peng et al. (2012) used probabilistic generative models for risk scoring schemes, ranging from the simple Naïve Bayes to advanced hierarchical mixture models. Their proposed methods computed a real risk score of Android applications based on the requested permissions, and differentiated between malware and benign applications. However, application developers tend to excessively declare permissions in a manifest file, requiring the method to rely on other criteria for higher detection and classification accuracy.

Blasing et al. (2010) proposed an Android Application Sandbox (AASandbox), which enables static and dynamic analysis on the Android platform. In the static analysis phase, AASandbox decompressed installation files and disassembled intended executable files,

then compared them with pre-defined malicious patterns. In the dynamic analysis phase, it hijacked system calls for logging and built a frequency table of system calls. However, the dynamic analysis methods based on the frequency of system calls need a more elaborate and redefined process in order to improve its detection or classification accuracy. The function name of the system call as well as arguments used in the system call need to be considered.

Yang et al. (2012) introduced a systematic approach, called Money-Guard, to detect stealthy money-stealing applications in the Android market. Money-Guard checked for API calls and billing-related permissions to detect stealthy money-stealing malware, but could not identify various malicious behavioral patterns except for malware sending premium-rate SMS.

Yan and Yin (2012) proposed DroidScope built on top of QEMU and enabled to reconstruct the OS-level and Java-level semantic views simultaneously. They analyzed malware by collecting native and Dalvik instruction traces, API-level activity, and information leakage. Reina et al. (2013) introduced CopperDroid, an approach built on top of QEMU to automatically perform dynamic analysis of Android malware. CopperDroid conducted a unified analysis to characterize low-level OS-specific and high-level Android-specific behaviors (e.g., information leakage, sending SMS) by observing and analyzing system call invocations, IPC and RPC interactions. Rastogi et al. (2013) proposed AppsPlayground, a framework for automatic dynamic analysis, which executes a suspicious application on emulator built on top of QEMU. AppsPlayground determined whether malicious activities were carried out or not by tracking information leakage and monitoring sensitive API and system calls. These approaches conducted fine-grained analysis, thus they may suffer from a high overhead. Moreover, they are needed to monitor synchronized low-level and high-level events. Andro-profiler also monitors low-level and high level events. Since our approach is implemented based on Droidbox, our approach is more stable since it only requires adding functionality related to hooking for low-level events.

Grace et al. (2012) proposed an anti-malware system, RiskRanker, to determine whether or not an application conducts malicious behavior by measuring potential security risk. RiskRanker classified an application into a high-risk application if it had exploit code for vulnerabilities in the OS. RiskRanker reported an application as a medium-risk application that enables to hijack sensitive information or subscribe premium service without victim's consent. Moreover, RiskRanker inspects malware embedding encryption and dynamic loading methods. However, RiskRanker mainly depends on signatures of malicious code to detect malware and fails to react to malware embedding obfuscation methods like Proguard. Wu et al. (2012) proposed DroidMat, which is a feature based malware detection method. DroidMat chose the requested permissions, Intent message, and API calls as feature vectors, extracted them from various resources such as manifest file and bytecode. By leveraging a *K*-means clustering algorithm, DroidMat modeled malware samples according to their characteristics, and then determine whether or not an application is malicious by leveraging *K*-NN algorithm. Arp et al. (2014) proposed DREBIN which utilizes the used permissions, suspicious API calls, and network addresses as feature vectors for identifying applications. DREBIN extracted those features from the manifest and `dex` bytecode files, and identified malware by leveraging

Support Vector Machine (SVM) algorithm. Zhou et al. (2012) proposed DroidRanger, which identifies malicious behavior through both permission-based behavioral footprint scheme for the detection of known malware and a heuristic-based filtering scheme for detection of 0-day malware. Weichselbaum et al. (2014) introduced Andrubis, which is an extension of Anubis (2011) for analyzing Android malware. Andrubis is an automatic analysis system coupled with static and dynamic methods. Vidas et al. (2014) presented A5, an automated anti-malware system based on static and dynamic analysis. Since DroidMat, DREBIN, DroidRanger, Andrubis, and A5 leveraged permission information of the manifest file as a feature, their accuracy is limited, just similar to other permission-based approaches; Kirin and Peng et al. (2012)'s work. Furthermore, they are ineffective in identifying benign applications because relevant rules only focus on detecting malware, thus produce large false alarms, despite other compensating rules being leveraged.

Spreitzenbarth et al. (2013) proposed Mobile-Sandbox, static and dynamic analyzer for Android applications, like in AASandbox. In the static analysis phase, Mobile-Sandbox parsed a manifest file, decompiled the application, and checked whether suspicious permissions are used or not. In the dynamic analysis phase, they executed the application on Droidbox, logged every operation of the application, and recorded native library calls executed by processes. They extracted native library calls by exploiting (ltrace 1997); ltrace is executed after installation process is completed.

Hybrid methods

In hybrid detection methods, clients collect meta information on applications on the device and send that information to a remote server. The remote server then analyzes this information using a detection algorithm and makes a decision on whether an application is benign or malicious. This approach compensates for the drawbacks of the online and offline detection methods. However, users have to agree in advance on what client module will send user information to the remote server.

Burguera et al. (2011) proposed a lightweight client called Crowdroid which monitored system calls, made a frequency table using those system calls, and sent them to a centralized server. The remote server then identified malicious behavior in a statistical manner and detected malware using a *K*-means clustering algorithm. Crowdroid extracted system calls by exploiting Strace (1991), but Strace is executed after installation—Crowdroid cannot detect malicious behavior during the installation process, and depends on the functionality of Strace. Isohara et al. (2011) proposed a kernel-based behavior analysis system that consisted of a system call log collector on an Android device and a log analyzer on a remote server. The client collected system calls generated at installation time and sent the logs to a remote server. The remote server then compared patterns in the logs with 16 pre-defined patterns. Since pre-defined behavior patterns mainly focused on malicious behaviors such as restricted information leakage, jailbreak, and abuse of root privileges, their system could not detect malicious behavior such as sending premium-rate SMS and calling premium-rate code. They also do not guarantee sufficient scalability.

Schmidt et al. (2009) proposed a collaboration mechanism for Android platform security comprising a log collector on the device and a remote analyzer. In their proposed system, the client monitored the behavior of the malicious application at the installation time, ran

analysis based on the similarity of the function call set used, exchanged the result of analysis with neighboring devices, and performed collaborative malware detection.

Kang et al. (2015) introduced a lightweight anti-malware system based on the serial number of a certificate and code similarity in an application. Based on similarity algorithm, they classified malware into the similar group, comparing the similarity of API sequence, permission usage, and system command usage. Jang et al. (2015) proposed a feature-rich anti-malware system based on based on similarity matching of malware-centric and malware creator-centric information. Their system classified malware samples into similar subgroups by exploiting the profiles extracted from integrated footprints, which are implicitly equivalent to distinct behavior characteristics. Dissimilar to our previous works (Kang et al. 2015; Jang et al. 2015), Andro-Profile analyzes malware characterization based on dynamic behavior analysis, coupled with Droidbox.

Behavior profiling

In the literature of traditional malware research related to personal computers operating Microsoft Windows, Bayer et al. (2009) proposed a method for scalable behavior-based malware clustering. The method contributes to the theoretical foundations of malware analysis by discussing the behavior-based profiling formally. Given the relevance of this work to our work, we review definitions of behavior profiling from the aforementioned work for the completeness of our presentation, and incorporate details specific to our proposed system in the following.

Definition *(behavior profiling)* A behavior profiling P is defined by four tuples as $P = (O, OP, \Gamma, \Delta)$, where O is the set of all objects and OP is the set of all operations, which is represented in nested dictionary form as {name : {target : attribute}}. $\Gamma \subseteq (O \times OP)$ is a relation assigning more than one operation to each other, and $\Delta \subseteq ((O \times OP), (O \times OP))$ represents the sequence-unrelated set, which is equivalent to integrated system logs.

Object An object represents an abstract functionality that malware samples need for carrying out the malicious behavior. Tam et al. (2015) manually analyzed many malware samples from various datasets such as contagion and Android malware Genome Project. They classified malicious behaviors into six groups according to behavior patterns: make call, send SMS, network access, access personal information, alter filesystem, and execute external application. We also manually inspected malware samples we had, and then defined malicious behavior as outlined by Tam et al. (2015); since some behavior patterns are not found in our dataset, we leave them out. We define malicious behavior as the sending of premium-rate SMS, the calling of premium-rate number, the sending of sensitive information, and converting data for transmission. We do not consider malicious behavior such as privilege escalation and Command and Control (C&C) attack since dynamic analysis methods hardly detect malware executing malicious behavior under given condition (e.g., SDK version, cellular network connection status, time, or place). We formally define object as following:

```
Object ::= Object-type
Object-type ::= Telephony|Phone|Network
```

Operation An operation represents a concrete malicious behavior. Formally, an operation comprises of operation-name, operation-target, and operation-attribute. Operation-name is the identifier for malicious behavior. Operation-target is the attack objective of malware, such as contents of external storage and system information. Operation-attribute is a meaningful value that the malware wants to obtain; for example, the attribute of country code (operation-target) is Korea, and operation-name is sending sensitive information. We formally define operation as follows:

```
Operation ::= {Operation-name : {Operation-target : Operation-attribute}}
Operation-name ::= Sending SMS|Calling|Sending sensitive information
                  |Converting data
Operation-target ::= Premium-rate SMS/number|deviceID|IMEI|IMSI|MCC|MNC|...|etc.
```

Table 2 shows an example of mapping of network object and the corresponding operations. In the case of malicious behavior for sending sensitive information, we represent the profile of that behavior as follows: "{Network : {Sending sensitive information : {{IMEI : 357242043237517}, {MCC : 310}, {MNC : 260}, {Location : GPS Coordinates } …, } } }".

Andro-profiler: an anti-malware system

In the following we review the design and operation of Andro-profiler, a hybrid system for malware analysis and classification that combines the on-device capabilities for profiling and off-device capabilities for analysis and classification.

Overview

As illustrated in Fig. 1, we propose a hybrid anti-malware system that consists of a client application on the mobile device and a profiling and analysis remote server. The client application on the mobile device collects installed application information, and sends that information to the remote server; the client application only sends application-specific information such as the hash digest of `apk` file and package name. If the remote server cannot crawl that application, the client application sends the application package

Table 2 Example of mapping of network object

Type	Name	Target	Attribute
Network	Sending sensitive information	Android Id	3531505c0b421c4d
		Device type	Android
		IMEI	357242043237517
		IMSI	310005123456789
		MCC	310
		MNC	260
		OS version	10
		SDK version	2.3.4
		Carrier	Android
		Country code	en
		Location	GPS coordinates
	Converting data	Cipher algorithm	No, DES, AES, Blowfish
		Destination URL	http://my365image.com
		Port	80
		Encoding algorithm	Gzip

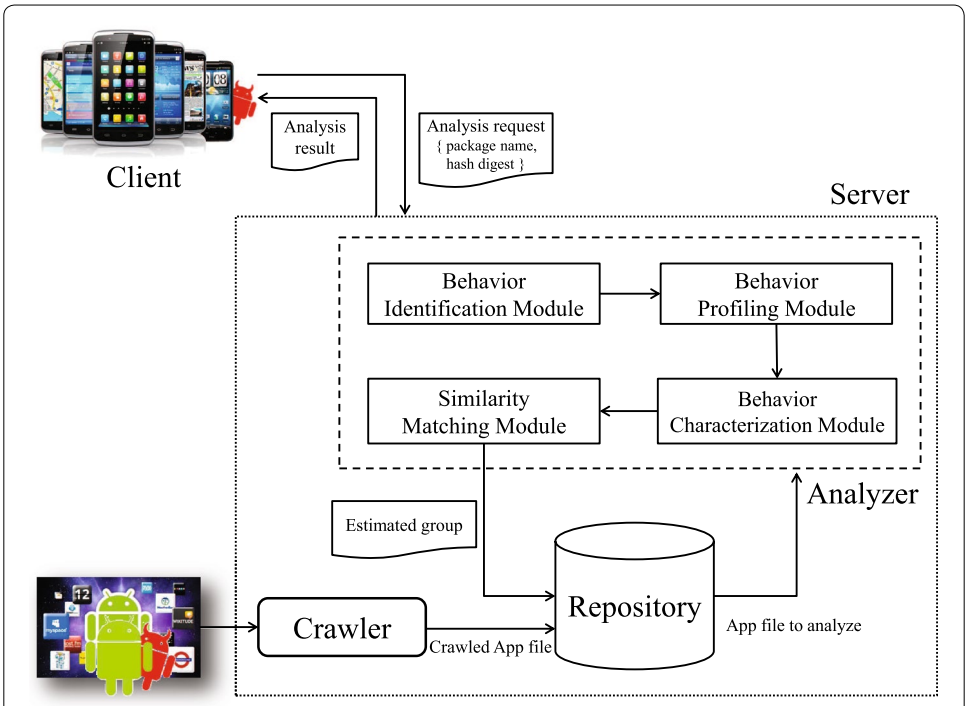

Fig. 1 Overall procedure of andro-profiler

file (`apk`) to the remote server. The remote server analyzes the malicious application and decides whether it is malicious or not based on its behavior. The remote server consists of three components: crawler, repository, and analyzer. The crawler component crawls applications from repositories, such as official markets and alternative markets. The crawled applications are then passed to the repository component which runs a duplication test by comparing the hash digest of the `apk` file to each other. If the crawled application is a duplicate, it is discarded; otherwise, the repository component sends that application to the analyzer component. After completing the analysis, the analyzer component sends the analysis results to both the repository component and the client application. Upon receiving the analysis results from the remote server, the client application displays the result on the screen to the user. The repository component searches its database upon the repository component receiving an analysis request from the client. If the repository component does not have analysis results to fulfill the client application's request, it fetches the crawler component. As illustrated in Fig. 2, the analyzer component has two processes: an extraction process of integrated system logs and a decision process. The extraction process of integrated system logs is composed of a behavior identification module, and the decision process is composed of three modules: a behavior profiling module, a behavior categorization module, and a similarity matching module. In the following, we review the extraction and decision processes.

Extraction process of integrated system logs

In following subsection, we review the extraction process of integrated footprints used for our profiling phase.

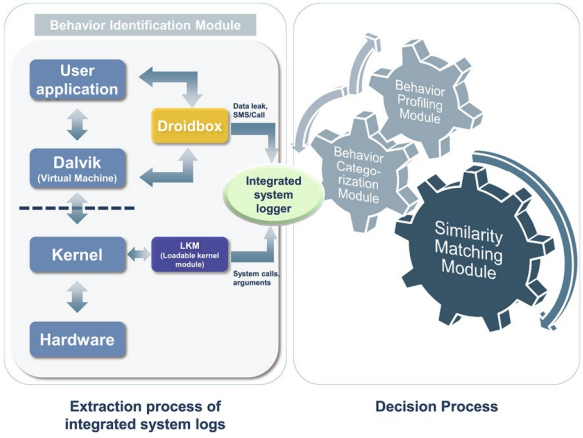

Fig. 2 Overview of the analyzer component

Behavior identification module

Andro-profiler conducts malware characterization based on dynamic behavior analysis. Our system extended Droidbox to embed the Loadable Kernel Module (LKM) for hijacking system calls including their arguments. More specifically, the Behavior Identification (BI) module in our system executes malware on an emulator and monitors malicious behavior in an isolated environment. Whenever malware is executed on the emulator, the BI fetches the integrated system logger. The integrated system logger parses system calls including their arguments provided by LKM and system logs provided by Droidbox; Droidbox monitors SMS, call, and network I/O. The parsed integrated system logs are then passed to the *decision process*.

Decision process

As shown in Fig. 2, the decision process consists of three modules: behavior profiling, behavior categorization, and similarity matching module. In the following we elaborate on each of those modules.

Behavior profiling module

The Behavior Profiling (BP) module parses the integrated system logs of a given application and makes the behavior profile. The BP module is implemented as described in previous section (Behavior Profiling). For example, the BP module makes a behavior profile of `GinMaster` which steals sensitive information, as illustrated in Fig. 3. According to the analysis report of F-Secure (F-Secure), `GinMaster` steals sensitive information, such as International Mobile Equipment Identity (IMEI), International Mobile Subscriber Identity (IMSI), User Identifier (UID), Subscriber Identification Module (SIM) number, telephone number, and network type, to a remote server. The behavior profile made by the BP module is similar to the analysis report of F-Secure, and it is simple and relatively easy to understand.

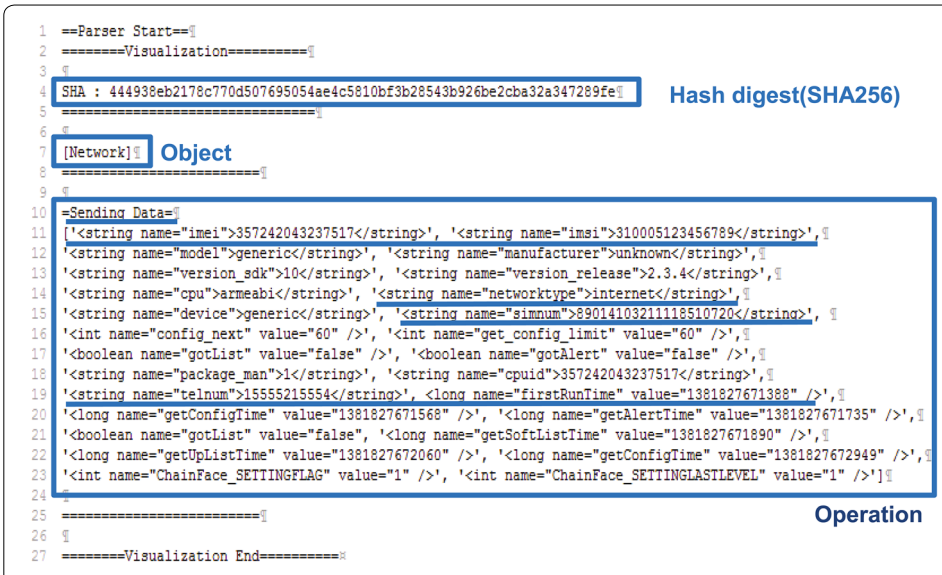

Fig. 3 Implementation of behavior profiling (e.g., GinMaster)

Behavior categorization module

The behavior categorization (BC) module categorizes a given application according to its behavior patterns. As we mentioned earlier, we define malicious behavior as the sending of premium-rate SMS, the calling of premium-rate number, the sending of sensitive information, and converting data for transmission. Since the numbers of malicious behavior patterns which we define are four, the possible permutation sets of malicious behavior patterns are 15 ($=\sum_{i=1}^{4} {}_4C_i$). If an application does not behave in accordance with a pre-defined malicious behavior, our system decides that the application is benign.

Similarity matching module

The different similarity metrics need to be applied to behavior factors since they have different types of argument. Instead of using machine learning approaches that usually use the same similarity metric for features, we design the appropriate similarity metrics for behavior factors. The similarity matching (SM) module computes the similarity score between the behavior profile of malicious application and representative behavior profile of each malware family. The SM then classifies the malicious application into the group with which it bears the most similarity based on its behavior. The representative behavior profile of each malware family has to depict the unique and common behavior patterns of each malware family, then SM module chooses one of the methods updating the representative behavior profile as follows:

1. Method 1: The first update method is *intersection*. The representative behavior profile for each malware family is updated by the intersection of behavior profiles of members in each subgroup. In the method 1 (intersection), and as the number of members of each malware family increases, the representative behavior profiles decrease.
2. Method 2: The second update method is *union*. The representative behavior profile for each malware family is updated by the union of behavior profiles of members in

each subgroup. In the method 2 (union), as the number of members of each malware family increases, the representative behavior profiles increase.

We define the similarity score as the intensity with which resources are accessed. Access to resources includes hardware resources (e.g., Call, SMS, Bluetooth, and Camera), system information, and private information (as detailed earlier); we define the similarity score as the weighted sum of the similarity of four behavior factors. The similarity score between the behavior profile of malicious application and a representative behavior profile for each malware family is given by:

$$S = \sum_i w_i \cdot BFS_i \quad where \quad \sum_i w_i = 1 \tag{1}$$

where BFS_i and w_i are the similarity and weight of behavior factor i, respectively. Similarity of behavior factor (BFS) is composed of four parts: similarity of sending premium-rate SMS (SS), calling premium-rate number (CS), sending sensitive information (SIS), and converting data (CDS). We choose the weight (w_i) to be 0.33 for SS, 0.33 for CS, 0.21 for SIS, and 0.13 for CDS—we determined that such settings for weight values are optimal and provide best performance through experiments.

Table 3 shows similarity metric to apply to each behavior factor, and we compute the similarity score for each behavior factor as follows:

1. We compute the similarity score for sending premium–rate SMS and calling premium-rate number, as comparing whether a relevant hardware resource is accessed or not. String similarity (e.g., phone number, code number) is less meaningful as a feature except for perfect matching since a difference of one bit yields the same result as with the difference of all bits in this case. Therefore, we give a similarity score of one if they have the same behavior; otherwise, we give a score of zero. Hence, the value of similarity score for both SS and CS is binary.
2. We compute the similarity score for sending sensitive information by applying the Jaccard index. We define the sensitive information as follows (highlighted in Table 2 by an example):

   (a) *System information* IMEI, IMSI, device ID, MCC, MNC, carrier name, device type, device model, OS version.
   (b) *Private information* external storage contents, location, country code, language.

Table 3 Similarity metric to apply to each behavior factor

Behavior factor	Behavior target	Similarity metric
Sending SMS	Premium-rate	Binary (0 or 1)
Calling	Premium-rate	Binary (0 or 1)
Sending sensitive information	System information, private information	Jaccard index [0, 1]
Converting data	Destination URL	Modified levenshtein distance [0, 1]
	Cipher algorithm (DES, AES, Blowfish)	Binary (0 or 1)
	Encoding algorithm (Gzip or not)	Binary (0 or 1)

We compute the similarity score for converting data (CDS), as the average of the similarity for a destination URL, cipher algorithm, and encoding algorithm. In the case of similarity of a destination URL, we first adopt the longest prefix matching. If a partial matching occurs, we adopt the Levenshtein distance to the residual string except the substring to which the longest prefix matching is used. For example, let A.B.C.D and A.B.E.F be two URLs. In this case, we adopt Levenshtein distance to the residual URLs: C.D and E.F. As for the cipher algorithm and encoding algorithm, we give a similarity score of one if they have used the same algorithm; otherwise, we give a score of zero. The value of similarity score for both SIS and CDS was [0, 1].

3. If a given application does not act maliciously (based on the defined criteria above) except for CDS, we consider that application to be benign.

Performance evaluation

In the following we demonstrate the performance and accuracy of Andro-profiler by highlighting aspects of implementation and testing it on various real-world mobile malware samples and families.

Implementation

Our anti-malware system is composed of a mobile device and a remote server; the client application is installed on the mobile device (SKY IM-A690S) running on the Android 2.3.3, and three components—a crawler, repository, and analyzer—were installed on the remote server. The remote server has an Intel(R) Xeon(R) X5660 processor and 4GB of RAM with 32-bit Ubuntu 12.04 LTS operating system; we performed all experiments in a hypervisor-based virtualization environment—VMWare ESXi; http://www.vmware.com/.

We implemented each component of our anti-malware system with Python high level programming language (as scripts) as follows:

1. The client component on the mobile device is implemented in the form of an application and communicated with the remote server. The crawler component sent the package name to GooglePlay and downloaded target application. The repository component stored the behavior profile of each application in a database.
2. The analyzer component is composed of the BI, BP, BC, and SM modules. In the following we provide details on each of those modules.

   (a) The BI module is implemented as python script coupled with Droidbox. The emulator is run on the Android 2.3.4 (level 10). In order to capture the malicious behavior, the BI module executed each application for 60 s after the installation process is completed. After capturing integrated system logs of malicious application, the BI module passed those logs to the BP module and restored the emulator to the initial state only for capturing malicious behavior.
   (b) The BP module parsed integrated system logs to make the behavior profile of each malware, and stored the behavior profile as a dictionary structure of the

Python language for efficient membership test. The parsing rule listed in Table 4 consists of system call and its arguments—only arguments provided by LKM, and information provided by Droidbox. The parsed behavior profile is encoded in a base-64 format and stored in database.

(c) The BC module categorized malicious application according to the behavioral patterns, and the SM module computed similarity score between behavior profile of malicious application and representative behavior profile for each family. The SM module classified a malicious application into the group with highest similarity score, which is at least 0.85. Whenever a new malware sample is queued into our anti-malware system for inspection, the SM module had continuously updated representative behavior profile according to the pre-chosen update method.

Experiment setup

For performance evaluation, 643 malware samples consisting of 5 malware families were collected from January 2013 to August 2013 through malware repositories such as VirusShare (2011), Contagio (2011). For 8840 benign samples, we crawled a variety of popular applications with high rankings (for the same periods) from GooglePlay. In

Table 4 Example of parsing rules for detecting malicious behavior

Behavior factor	Parsing rule	Comment
Sending SMS	**mms.transaction.SmsReceiverService**	**SMS**
Calling	access(/system/app/Phone.apk ~)	
	writev(3, OutgoingCallBroadcaster ~)	Calling
Sending sensitive information	open(/proc/cpuinfo ~), write(1, Processor ~)	CPU Spec.
	open(/sdcard ~), stat64(/sdcard/~)	Storage access
	stat64(/system/app/MediaProvider.apk),	
	access(/data/~/com.android.providers.media/databases),	
	com.android.providers.media.MediaScannerService),	
	open(/data/dalvik-cache/system@app @MediaProvider.apk@classes.dex)	Media file
	{stat64 | open | access}(/system/app/Contacts.apk),	
	{stat64 | open} (/data/~ @Contacts.apk@classes.dex)	Contact information
	⟨map⟩ ~ { NET_OP | mcc | mnc } ~ ⟨\map⟩, ⟨map⟩ ~ { networkOperator | sim_operator } ~ ⟨\map⟩	MCC, MNC
	⟨map⟩ ~ { affid | did | device_id | andide } ~ ⟨\map⟩	Device ID
	⟨map⟩ ~ { osversion | device_type } ~ ⟨\map⟩	OS version
	⟨map⟩ ~ { manufacturer | phoneModel | device_name | model } ~ ⟨\map⟩	Device
	⟨map⟩ ~ { network | wifi } ~ ⟨\map⟩	Wifi information
	⟨map⟩ ~ { carrier | device_carrier } ~ ⟨\map⟩	Carrier
	⟨map⟩ ~ { imei | imsi } ~ ⟨\map⟩	IMEI, IMSI
	⟨map⟩ ~ { longitude | latitude } ~ ⟨\map⟩	Location
	⟨map⟩ ~ { location | country_code | locale } ~ ⟨\map⟩	Country code
	⟨map⟩ ~ { language } ~ ⟨\map⟩	Language
Converting data	{sendto | OpenNet | SendNet | DataLeak} (~ Content-Encoding: gzip ~)	Encoding algorithm
	{sendto | OpenNet | SendNet | DataLeak}(~ CryptoUsage: {DES|AES|Blowfish} ~)	Cipher algorithm

the real world, malware comprises a small fraction of all android apps, so it makes sense to use a larger set of benign samples to mimic the realistic scenario. Duplicated samples were eliminated according to SHA 256. We also excluded malware samples diagnosed by fewer than 9 AV vendors included by the VirusTotal dataset (2004). We used textual description of malware produced by F-Secure (1999). The description of the samples is summarized in Table 5. In addition, the whole experimental results are available at http://ocslab.hksecurity.net/andro-profiler.

For the validation of our work, we used fivefold cross-validation to evaluate the performance in our experiments. The *k*-fold cross-validation is a widely used technique in machine learning. In a nutshell, the method partitions the dataset into *k* equal size subsets, where each subset is used only once for testing and validation of the training model, and the $k-1$ remaining subsets are used for training the model. This is, a model is built using $k-1$ subsets, and tested using the remaining subset. Then, the subset used in the previous step for testing is used for training, and a subset in the $k-1$ sets previously not used for testing is used for testing. The process is repeated *k* times by alternating the testing set, and the results are averaged over the runs.

Comparing different methods

To the best of our knowledge, the closest approaches in the literature to Andro-profiler are Crowdroid (Burguera et al. 2011), CopperDroid (Reina et al. 2013), and AppsPlayground (Rastogi et al. 2013). Crowdroid monitored invoked system calls and made frequency table of system calls at the client side. Crowdroid identified malicious behavior and detected malware utilizing the *K*-means algorithm at the server side. CopperDroid conducted automatic dynamic analysis to characterize low and high level behaviors by tracking system call invocation, IPC and RPC interactions. AppPlayground also conducted automatic dynamic analysis, and determined whether malicious behaviors were carried out by tracking information leakage and monitoring sensitive API and system calls. For completeness of our approach, we need to compare ours with these approaches. However, these approaches are not available for public use. Among them, it is quite straightforward to implement Crowdroid hooking system calls; it is impossible to implement other works because these approaches do not provide more detailed explanation. To conduct more fair performance evaluation and comparison, we make both systems work in a similar context and using similar settings: we modify Crowdroid to hook all system calls invoked during the execution processes, including the installation phase.

Table 5 Malware samples and benign samples for experiments

Category	Family	Quantity	Behavioral characteristics
Malware (643)	AdWo	401	Collect the sensitive information
	AirPush	60	Send SMS and collect the sensitive information
	FakeBattScar	44	Collect the sensitive information
	Boxer	42	Send SMS and collect the sensitive information
	GinMaster	96	Collect the sensitive information
Benign (8840)	Application	7164	Normal application
	Game	1676	Normal game application

Selection of weight for behavior factors

Andro-profiler needs to select appropriate weights (w_i) in order to guarantee the best performance. However, we cannot obtain a unique solution of Eq. (1) analytically, because there are only two equations given in order to compute values of four variables, which means that we cannot obtain an optimal solution of Eq. (1). We might obtain local optimum values of Eq. (1) through simple numerical approach (iterative method) as follows. First, we setup initial values of weight by solving arithmetic mean of them. We apply those values to the Eq. (1), then evaluate the classification capability. Next, we increase the weight of SS and CS, and decrease the weight of SIS and CDS. We then apply those values to the Eq. (1), and conduct the evaluation of classification capability iteratively. The reason we determine that the weight of CDS is smaller than other factors is as follows. First, if a client cannot connect to the remote server, a malware sample does not need to convert format of data for transmitting sensitive information. Second, benign applications also need an encoding algorithm for efficient transmission and cipher algorithm for secure communication. We adjust the weight of SIS in order to maximize the effect of calling and sending premium-rate SMS.

We proceed with the iterative steps until the tendency of classification accuracy is changed. We believe that our system reaches a local optimum at that point. Table 6 shows that the results of simple numerical approach according to weight change. We choose the value of the weight (w_i) to be 0.33 for SS, 0.33 for CS, 0.21 for SIS, and 0.13 for CDS, since it provides a good performance that matches close to the ground truth.

Experiment results and analysis

Our performance evaluation focuses on the effectiveness of malware classification, discriminatory ability between malware and benign applications, and the efficiency of malware classification. We demonstrate that our system performs well in detecting and classifying malware families. We used the accuracy, false positive, and false negative as the performance metric, since the metric for performance evaluation must focus on the predictive capability of the model. We measured the accuracy as the total number of the hits of the classifier divided by the number of instances in the whole dataset. The performance of malware classification model is determined by how well the model detects and classifies various pieces of malware. Moreover we used the Receiver Operating

Table 6 The classification accuracy and the number of cluster according to changes of weight (e.g., Method 1)

No	Weight of behavior factor				Number of clusters		Accuracy
	SS	CS	SIS	CDS	Malware	Benign	
1	0.25	0.25	0.25	0.25	8	4	0.98
2	0.27	0.27	0.24	0.22	6	2	0.98
3	0.29	0.29	0.23	0.19	6	2	0.98
4	0.31	0.31	0.22	0.16	6	2	0.98
5	*0.33*	*0.33*	*0.21*	*0.13*	*6*	*1*	*0.98*
6	*0.35*	*0.35*	*0.20*	*0.10*	*6*	*1*	*0.98*

The number of clusters means that the number of groups that malware/benign samples are classified into. Italic text means that the tendency of classification accuracy is changed. At this point, we believe, our system reaches a local optimum for the best performance

Characteristic (ROC) curve as the method for comparing classification models. To compare the ROC performance of classifiers intuitively, we calculated the area under the curve (AUC; also known as the integral) of each classifier, since the AUC represents the ROC performance in a single scalar value (Fawcett 2006).

Effectiveness of malware classification

First, we demonstrate that our proposed method provides effective metric to detect and classify malware families. Table 7 presents the result of similarity comparison with the representative profile of each malware family and benign applications. Despite that `Boxer` sends premium-rate SMS according to anti-virus (AV) analysis report, our emulator-based approach fails to capture sending premium-rate SMS due to connection error; our method only captures sending sensitive information. However, our system performs well in classifying all malware including `Boxer`. Since the difference of similarity score among all malware is smaller than the threshold (0.85), that can be good metric for detecting and classifying malware. The difference of similarity score for `AirPush` is much larger than the others, because `AirPush` sends premium-rate SMS and sends sensitive information while the other malware families send sensitive information. Since benign applications do not act maliciously, it is natural that the difference of similarity score between malware and benign applications is large based on the metrics and features utilized for computing the behavior profile.

Next, Table 8 shows that Andro-profiler performs well in classifying malware families with 100 % classification accuracy on average, regardless of the update method. Furthermore, Andro-profiler is shown to outperform Crowdroid, which gives an average classification accuracy of 49 %. Some factors may have affected that Crowdroid underperforms

Table 7 The similarity comparison with representative behavior profile of each malware family and benign

Similarity	AdWo	AirPush	Boxer	FakeBattScar	GinMaster
AdWo	–	0.37	0.70	0.70	0.70
AirPush	0.37	–	0.46	0.46	0.50
Boxer	0.70	0.46	–	0.79	0.79
FakeBattScar	0.70	0.46	0.79	–	0.79
GinMaster	0.70	0.50	0.79	0.79	–
Benign	0.04	0.13	0.13	0.13	0.13

Table 8 Classification performance for 643 malware

Category	Accuracy			AUC		
	Method 1	Method 2	Crowdroid	Method 1	Method 2	Crowdroid
Malware						
AdWo	*1.00*	*1.00*	0.83	*1.00*	*1.00*	0.73
AirPush	*1.00*	*1.00*	0.02	*1.00*	*1.00*	0.51
Boxer	*1.00*	*1.00*	0.37	*1.00*	*1.00*	0.63
FakeBattScar	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*
GinMaster	*1.00*	*1.00*	0.22	*1.00*	*1.00*	0.54
Average	*1.00*	*1.00*	0.49	*1.00*	*1.00*	0.68

Italic text means that Andro-profiler outperforms Crowdroid in classifying malware families

the Andro-profiler. Since invoked system calls among malware families are similar to each other, Crowdroid limits to classify malware families; malware families mainly call out system calls (e.g., read(), close(), open(), write(), recvmsg()). Since FakeBattScar calls out more system calls (e.g., open(), close()) than others and Adwo calls out system call of read() constantly, two malware families can be classified well. Furthermore, Andro-profiler gives 47 % performance improvement advantage over Crowdroid in terms of the AUC. In the case of method 1, our system clusters `Airpush` samples into two groups. We conducted a deep analysis to understand the reason method 1 of our system clustered such samples into two groups, and found that almost half of Airpush samples sent premium-rate SMS and collected sensitive information (e.g., IMEI, Android version, location information, and carrier), whereas the other half only collected sensitive information. To this end, we found that our system identified malicious behavior and classified malware according to behavior patterns of malware families.

Discriminatory ability between malware and benign

When designing an anti-malware system, one important factor which we should also consider is its discriminatory ability between malware and benign applications. Anti-malware systems must detect malware with small errors in terms of false positive and false negative. We believe that it is more important for an anti-malware system to detect malware with small false negative than false positive. However, for commercial reasons, one may think the opposite: users can be bothered if their benign applications are misclassified as malware. Table 9 shows that Andro-profiler performs well in detecting and classifying malware families with 98 % classification accuracy on average, regardless of the update method, while Crowdroid detects malware families with 90 % classification accuracy on average. Some factors may have affected that Crowdroid underperforms the Andro-profiler. Since invoked system calls between malware and benign samples are similar to each other, Crowdroid limits to detect and classify malware families; all samples mainly call out system calls (e.g., read(), close(), open(), write(), recvmsg()). Among these, FakeBattScar calls out more system calls (e.g., open(), close()) than others and other malware families have similar call frequencies to benign

Table 9 Classification performance for 643 malware and 8840 benign samples

Category	Accuracy			AUC		
	Method 1	Method 2	Crowdroid	Method 1	Method 2	Crowdroid
Malware						
AdWo	*1.00*	*1.00*	0.01	*1.00*	*1.00*	0.49
AirPush	*1.00*	*1.00*	0.00	*1.00*	*1.00*	0.50
Boxer	*1.00*	*1.00*	0.00	*1.00*	*1.00*	0.50
FakeBattScar	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*
GinMaster	*1.00*	*1.00*	0.00	*1.00*	*1.00*	0.49
Benign	*0.97*	*0.97*	0.96	*0.99*	*0.99*	0.52
Average	*0.98*	*0.98*	0.90	*0.99*	*0.99*	0.52

Italic text means that Andro-profiler outperforms Crowdroid in detecting malware and classifying malware families

samples, then malware families except for FakeBattScar cannot be detected and classified well. Our proposed methods also outperform Crowdroid by improving its AUC by about 90 %. Table 10 shows that our system performs well in detecting and classifying malware families with about 3 false positives and 38 false negatives, while Crowdroid detects and classifies malware families with over 100 false positives and false negatives. All malware families except Airpush were classified with low false positives rate and false negative rate.

Andro-profiler misclassified 225 benign samples as malware. We conducted a deep analysis to understand the high false positives with Andro-profiler. Interestingly, we found that some benign samples collected user's sensitive information, which we defined as a trigger for classifying malicious applications (e.g., IMEI, device ID, UUID, latitude, and longitude). To understand whether other anti-malware systems and scanners considered those benign applications as malware or not, we uploaded those suspected GooglePlay samples to VirusTotal and checked scanning results of various anti-virus vendors. As a result, we found that 110 out of the suspicious benign samples (accounting for about 49 %) were diagnosed as malware. The high rate of misclassification of benign applications is, however, understandable given various potential reasons for such infiltration of gray area applications into the market place (Krebs 2013).

Effectiveness of detecting 0-day malware

We demonstrate the effectiveness of detecting 0-day malware detection. We define an application as a 0-day malware if it has malicious behavior and it cannot be detected by AV vendors. In order to verify that we had appropriately detected 0-day malware, we made 91 variant samples consisting of Adwo and AirPush families by leveraging ADAM (Zheng et al. 2013). All samples used as the base application for the variants are among the ones which are used in the previous experiments, and detected by VirusTotal as malware samples. After creating the variants, we uploaded them (as samples) to the VirusTotal, and checked scanning results of various anti-virus (AV) vendors such as F-Secure, Kaspersky, ClamAV, and Avast. We noted that none of the submitted samples is reported as a malware when we carried out our experiment. As a result of our experiment using Andro-profiler, we found that it performed well in

Table 10 Classification performance for 643 malware and 8840 benign samples

Category	Method 1		Method 2		Crowdroid	
	FPs	FNs	FPs	FNs	FPs	FNs
Malware						
AdWo	0	0	0	0	229	397
AirPush	17	0	17	0	0	60
Boxer	0	0	0	0	0	42
FakeBattScar	0	0	0	0	2	0
GinMaster	0	0	0	0	0	96
Benign	0	225	0	225	589	461
Average	2.83	37.50	2.83	37.50	136.67	176

FPs and FNs refer to false positives and false negatives

detecting all of the variant malware samples with 100 % classification accuracy on average, regardless of the update method.

Efficiency of malware classification

Our proposed system only takes 55 s/MB for classifying each malware; we exclude setup time for analysis such as booting time of emulator. The majority of this time is spent in making the behavior profile; it takes only 0.2 s on average to classify malware into each family.

While the performance of our system is operationally reasonable, our system is scalable both horizontally and vertically by design. Horizontally, and given that our server side components are run in a virtual environment, one can fork multiple servers by utilizing multiple virtual machines that exploit the multi-core nature of today's commodity computers. Vertically, our system can benefit from being developed in a lower level language, such as the C language, which would make the classification process run faster.

Limitations

Andro-profiler has a few limitations for detecting and classifying malware, since our proposed method uses integrated system logs as a feature vector and employs dynamic analysis techniques to capture malware's behavior. First, it is difficult for our system to analyze malware that are executed only under given conditions (e.g., SDK version, cellular network connection status, time, or place). However, this shortcoming is addressable by having various platforms tailored with various settings, as used for traditional malware in Mohaisen et al. (2013). It is also impossible for our system to analyze malware embedding anti-malware analysis techniques. Second, our emulator-based anti-malware system is dependent on SDK version of emulator, so our approach has limitation on analyzing malicious behavior related to privilege escalation. However, those are common drawbacks of dynamic analysis method or emulator-based detection method and addressed in the literature at some expense.

Finally, our approach analyzes malware on an emulator without interaction between human and device: autonomous installation and execution. When a malware behave upon an update or by utilizing a drive-by download attack (Zhou and Jiang 2012), our approach is limited in reacting to such malware. However, autonomous installation and execution is an inevitable procedure for automation of dynamic analysis. Depending on the number of malware samples to be analyzed, one can adopt manual human interactions to analyze malware samples and vet the outcomes of the automatic classification procedure, as used in Mohaisen et al. (2013).

Conclusion and future work

In this paper, we have presented Andro-profiler, an anti-malware system based on behavior profiling. Using Andro-profiler, we classified malware by exploiting the behavior profiling extracted from integrated system logs, which are implicitly equivalent to distinct behavior characteristics. Our behavior profiling is simple and relatively easy to understand, whereas Andro-profiler is capable of distinguishing benign and malicious applications, and malicious applications into families. Furthermore, Andro-profiler is capable of detecting 0-day threats, which are missed by antivirus scanners.

Our experiments demonstrate that Andro-profiler performs well in detecting and classifying malware families with over 98 % classification accuracy on average regardless of update method while Crowdroid, a closely related work from the literature, performs under 90 % classification accuracy on average. Our experiment results indicate that it takes 55 s/MB to analyze a malware on average, with a lot of opportunities for improvements on scalability. Our system hence enables AV vendors to react to many species of malicious samples by classifying and matching these with previous ones effectively and efficiently.

There are several directions that we will pursue in the future. First, we would like to augment our system to not only rely on dynamic and behavioral features, but also static features that are easy to obtain from the applications at scale. Furthermore, we will explore scalability issues associated with our system by implementing some of the guidelines noted in section "Efficiency of malware classification".

Authors' contributions

JJ carried out conception generation and experimental design, acquisition of data, analysis and interpretation of data, and drafting the manuscript. JY carried out acquisition of data and experimental design, and implementation of our prototype. AM, JW, and HKK carried out the concept generation and interpretation of data, revising the manuscript. All authors read and approved the final manuscript.

Author details

[1] Graduate School of Information Security, Korea University, Seoul, Republic of Korea. [2] Computer Science and Engineering Department, State University of New York at Buffalo (SUNY Buffalo), Buffalo, NY, USA.

Acknowledgements

This work was supported by the ICT R&D Program of MSIP/IITP. [14-912-06-002, The Development of Script-based Cyber Attack Protection Technology]. A two-page abstract on this work appeared in Jang et al. (2014). The work proposed in this paper significantly enhances the prior work, technically and content-wise, including the motivation, related-work, design, and evaluation.

Competing interests

The authors declare that they have no competing interests.

Received: 5 November 2015 Accepted: 16 February 2016
Published online: 03 March 2016

References

Anubis (2011) Anubis—malware analysis for unknown binaries. https://anubis.iseclab.org/

Arp D, Spreitzenbarth M, Hübner M, Gascon H, Rieck K, Siemens C (2014) Drebin: effective and explainable detection of Android malware in your pocket. In: Proceedings of the 21th annual network and distributed system security symposium (NDSS'14)

Bayer U, Comparetti P, Hlauschek C, Kruegel C, Kirda E (2009) Scalable, behavior-based malware clustering. In: Proceedings of the 16th annual network and distributed system security symposium (NDSS'09)

Blasing T, Batyuk L, Schmidt AD, Camtepe S, Albayrak S (2010) An Android application sandbox system for suspicious software detection. In: 2010 5th international conference on malicious and unwanted software (MALWARE), pp 55–62

Bose A, Hu X, Shin KG, Park T (2008) Behavioral detection of malware on mobile handsets. In: Proceedings of the 6th international conference on mobile systems, applications, and services, MobiSys'08, pp 225–238

Bugiel S, Davi L, Dmitrienko A, Fischer T, Sadeghi AR, Shastry B (2012) Towards taming privilege-escalation attacks on Android. In: Proceedings of the 19th annual symposium on network and distributed system security

Burguera I, Zurutuza U, Nadjm-Tehrani S (2011) Crowdroid: behavior-based malware detection system for Android. In: Proceedings of the 1st ACM workshop on security and privacy in smartphones and mobile devices, SPSM'11, pp 15–26

Contagio (2011) Contagio mobile–mobile malware mini dump. http://contagiominidump.blogspot.kr/. Accessed 28 Oct 2015

Droidbox (2011) Droidbox—Android application sandbox—Google project hosting. https://code.google.com/archive/p/droidbox/. Accessed 28 Oct 2015

Enck W, Gilbert P, Chun BG, Cox LP, Jung J, McDaniel P, Sheth AN (2010) TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In: Proceedings of the 9th USENIX conference on operating systems design and implementation, OSDI'10, pp 1–6

Enck W, Ongtang M, McDaniel P (2009) On lightweight mobile phone application certification. In: Proceedings of the 16th ACM conference on computer and communications Security, CCS'09, pp 235–245

F-Secure. https://www.f-secure.com/v-descs/trojan_android_ginmaster.shtml. Accessed 28 Oct 2015

F-Secure (1999) F-secure, 25 years of the best protection in the world. https://www.f-secure.com/en/welcome. Accessed 28 Oct 2015
Fawcett T (2006) An introduction to ROC analysis. Pattern Recognit Lett 27(8):861–874
Grace M, Zhou Y, Zhang Q, Zou S, Jiang X (2012) Riskranker: scalable and accurate zero-day Android malware detection. In: Proceedings of the 10th international conference on mobile systems, applications, and services, MobiSys'12, pp 281–294
Isohara T, Takemori K, Kubota A (2011) Kernel-based behavior analysis for Android malware detection. In: 2011 seventh international conference on computational intelligence and security (CIS), pp 1011–1015
Jang J, Yun J, Woo J, Kim HK (2014) Andro-profiler: anti-malware system based on behavior profiling of mobile malware. In: 23rd international World Wide Web conference, WWW'14, Seoul, Republic of Korea, April 7–11, 2014, companion volume, pp 737–738. doi:10.1145/2567948.2579366
Jang JW, Kang H, Woo J, Mohaisen A, Kim HK (2015) Andro-AutoPsy: anti-malware system based on similarity matching of malware and malware creator-centric information. Digit Invest 14:17–35
Kang H, Jang JW, Mohaisen A, Kim HK (2015) Detecting and classifying Android malware using static analysis along with creator information. Int J Distrib Sens Netw 2015. doi:10.1155/2015/479174. Article ID 479174
Krebs B (2013) Mobile malcoders pay to (Google) Play. Krebs on security. http://bit.ly/1kranE5
ltrace (1997) ltrace. http://ltrace.org/. Accessed 28 Oct 2015
McAfee (2015) McAfee labs threat s report, August 2015. http://www.mcafee.com/us/resources/reports/rp-quarterly-threats-aug-2015.pdf. Accessed 28 Oct 2015
Mohaisen A, Alrawi O, Larson M (2013) Amal: highfidelity, behavior-based automated malware analysis and classification. Tech. rep., Verisign Labs, Tech. Rep
Mohaisen A, West AG, Mankin A, Alrawi O (2014) Chatter: classifying malware families using system event ordering. In: IEEE conference on communications and network security, CNS 2014, San Francisco, CA, USA, October 29–31, 2014, pp 283–291. doi:10.1109/CNS.2014.6997496
Pearce P, Felt AP, Nunez G, Wagner D (2012) AdDroid: privilege separation for applications and advertisers in Android. In: Proceedings of the 7th ACM symposium on information, computer and communications security, ASIACCS'12, pp 71–72
Peng H, Gates C, Sarma B, Li N, Qi Y, Potharaju R, Nita-Rotaru C, Molloy I (2012) Using probabilistic generative models for ranking risks of Android apps. In: Proceedings of the 2012 ACM conference on computer and communications security, CCS'12, pp 241–252
Rastogi V, Chen Y, Enck W (2013) AppsPlayground: automatic security analysis of smartphone applications. In: Proceedings of the third ACM conference on data and application security and privacy, CODASPY'13, pp 209–220
Reina A, Fattori A, Cavallaro L (2013) A system call-centric analysis and stimulation technique to automatically reconstruct Android malware behaviors. In: Proceedings of the 6th European workshop on system security (EUROSEC). Prague, Czech Republic
Schmidt AD, Bye R, Schmidt HG, Clausen J, Kiraz O, Yuksel K, Camtepe S, Albayrak S (2009) Static analysis of executables for collaborative malware detection on Android. In: IEEE international conference on communications, 2009. ICC'09, pp 1–5
Shabtai A, Elovici Y (2010) Applying behavioral detection on Android-based devices, pp 235–249
Spreitzenbarth M, Freiling F, Echtler F, Schreck T, Hoffmann J (2013) Mobile-sandbox: having a deeper look into Android applications. In: Proceedings of the 28th annual ACM symposium on applied computing, SAC'13, pp 1808–1815
Strace (1991) Strace—useful diagnostic, instructional, and debugging tool. http://sourceforge.net/projects/strace/. Accessed 28 Oct 2015
Tam K, Khan SJ, Fattori A, Cavallaro L (2015) CopperDroid: automatic reconstruction of Android malware behaviors. In: 22nd annual network and distributed system security symposium, NDSS 2015, San Diego, California, USA, February 8–11, 2015
Vidas T, Tan J, Nahata J, Tan CL, Christin N, Tague P (2014) A5: automated analysis of adversarial android applications. In: Proceedings of the 4th ACM workshop on security and privacy in smartphones and mobile devices. ACM, pp 39–50
VirusShare (2011) VirusShare.com-Because Sharing is caring. http://virusshare.com/. Accessed 28 Oct 2015
VirusTotal (2004) VirusTotal—free online virus, malware and URL scanner. https://www.virustotal.com/en/. Accessed 28 Oct 2015
Weichselbaum L, Neugschwandtner M, Lindorfer M, Fratantonio Y, van der Veen V, Platzer C (2014) Andrubis: Android malware under the magnifying glass. Tech. Rep. TRISECLAB-0414-001, Vienna University of Technology
Wu DJ, Mao CH, Wei TE, Lee HM, Wu KP (2012) Droidmat: Android malware detection through manifest and api calls tracing. In: Proceedings of the 2012 seventh Asia joint conference on information security, ASIAJCIS'12, pp 62–69
Yan LK, Yin H (2012) DroidScope: seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis. In: Proceedings of the 21st USENIX conference on security symposium, Security'12, pp 29–29
Yang C, Yegneswaran V, Porras P, Gu G (2012) Detecting Money-stealing apps in alternative Android markets. In: Proceedings of the 2012 ACM conference on computer and communications security, CCS'12, pp 1034–1036
Zheng M, Lee PPC, Lui JCS (2013) ADAM: an automatic and extensible platform to stress test Android anti-virus systems. In: Proceedings of the 9th international conference on detection of intrusions and malware, and vulnerability assessment, DIMVA'12, pp 82–101
Zhou Y, Jiang X (2012) Dissecting Android malware: characterization and evolution. In: 2012 IEEE symposium on security and privacy (SP), pp 95–109
Zhou Y, Wang Z, Zhou W, Jiang X (2012) Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In: Proceedings of the 19th annual network and distributed system security symposium