Monitoring Android for Collaborative Anomaly Detection: A First Architectural Draft

Technical Report: TUB-DAI 08/08-02

Aubrey-Derrick Schmidt, Rainer Bye, Hans-Gunther Schmidt, Kamer Ali Yüksel, Osman Kiraz, Jan Clausen, Karsten Raddatz, Ahmet Camtepe, and Sahin Albayrak

August 21, 2008
## Contents

1 Introduction .................................................. 4

2 Related Work ................................................ 5
   2.1 Mobile Device Intrusion Detection .......................... 5
   2.2 Intrusion Detection Systems in Ad-Hoc Networks .......... 7

3 Overall System Architecture .................................. 7

4 Monitoring and Detection Client Architecture ............... 8
   4.1 Linux Operating System Level ............................. 8
   4.2 Linux Application Level .................................. 9
   4.3 Java Application Level .................................. 10

5 General Detection Architecture ................................. 11
   5.1 Detection Mechanism ..................................... 11
   5.2 Server supported learning ................................. 13
   5.3 Collaborative Intrusion Detection in Ad-Hoc Network Scenario 13

6 Conclusion and Future Work .................................. 14

7 Acknowledgements ............................................. 14

References ....................................................... 14

### List of Figures

1 Overall System Architecture .................................. 8
2 Monitoring and Detection Client Architecture ............... 9
3 Architecture of Detection Mechanism .......................... 12

### List of Tables
1 Introduction

Our daily lives become more and more dependent upon smartphones due to their increased capabilities. Smartphones are used in various ways, e.g. for payment systems or assisting the lives of elderly or disabled people. Security threats for these devices become more and more dangerous since there is still a lack of proper security tools for protection. Android emerges as an open smartphone platform which allows modification even on operating system level and where third-party developers first time have the opportunity to develop kernel-based low-level security tools.

Android quickly gained its popularity among smartphone developers and even beyond since it bases on Java on top of "open" Linux in comparison to former proprietary platforms which have very restrictive SDKs and corresponding APIs. Symbian OS, holding the greatest market share among all smartphone OSs, was even closing critical APIs to common developers and introduced application certification. This was done since this OS was the main target for smartphone malwares in the past. In fact, more than 290 malwares designed for Symbian OS appeared from July 2004 to July 2008. Android, in turn, promises to be completely open source. Together with the Linux-based smartphone OS OpenMoko, open smartphone platforms may attract malware writers for creating malicious applications endangering the critical smartphone applications and owners’ privacy.

Since signature-based approaches mainly detect known malwares, anomaly-based approaches can be a valuable addition to these systems. They base on mathematical algorithms processing data that describe the state of a certain device. For gaining this data, a monitoring client is needed that has to extract “usable” information (features) from the monitored system. Our approach follows a dual system for analyzing these features. On the one hand, functionality for on-device light-weight detection is provided. But since most algorithms are resource exhaustive, remote feature analysis is provided on the other hand. Having this dual system enables event-based detection that can react to the current detection need. In our ongoing research we aim to investigates the feasibility of light-weight on-device detection for certain occasions. On other occasions, whenever significant changes are detected on the device, the system can trigger remote detection with heavy-weight algorithms for better detection results. In the absence of the server respectively as a supplementary approach, we also consider a collaborative scenario. Here, mobile devices sharing a common objective are enabled by a collaboration module to share information, such as intrusion detection data and results. This is based on an ad-hoc network mode that can be provided by a WiFi or Bluetooth adapter nearly every smartphone possesses.

This report is structured as follows. In Section 2, related work to mobile device-based intrusion detection with a focus on Ad-Hoc networks is presented. Section 3 introduces a simple global system architecture. In Section 4, the structure of the monitoring and detection client for mobile devices is presented. Section 5 describes the detection architecture of the system. Finally in Section 6, we conclude.
2 Related Work

Several efforts made by researchers to counter the expected outcome of attacks on smartphones. The early attempts started by adapting existing security solutions developed for PCs to mobile devices. Then, mobile specific light-weight intrusion detection systems started to be designed due to resource constraints of mobile devices in terms of memory and power. Furthermore, there were attempts to enhance these architectures using collaborative schemes. Especially use of mobile ad-hoc networks has made researchers to put effort on collaborative approaches. Since intrusion detection is quite a large research area, we will focus on IDS suitable for mobile and ad-hoc networks in this work.

2.1 Mobile Device Intrusion Detection

Guo et al. [1] describes possible attacks for smart-phones, which range from privacy violation and identity theft to emergency call center DDoS attacks and national crises, concerning their interoperability between the telecom networks and the Internet. They also describe defense solution space including smart-phone hardening approaches, Internet-side defense, telecom-side defense, and coordination mechanisms that may be needed between the Internet and telecom networks. They intend to layout the landscape and solution space rather then proposing full-fledged solutions.

Venugopal et al. [2] outline the considerations for malware detection on mobile devices and propose a signature based malware detection method that is well suited for use in mobile device scanning due to its low memory requirements and high scanning speed. Later, they compare their signature matching algorithm with Clam-AV\textsuperscript{1} scanner to evaluate its efficiency. However, their method is not able to detect new malware or variants of existing malware whose signatures are not in the database; thus, needs frequent updates which causes higher battery consumption.

Nash et al. [3] designed an IDS against a unique form of DoS attack known as a battery exhaustion attack by taking into account the performance, energy, and memory constraints of mobile devices. Their IDS uses several parameters, such as CPU load and disk accesses, to estimate the power consumption per process using a linear regression model, to identify processes that might be battery exhaustion attacks. Using energy-usage-based detection system have the advantage that they only need to monitor one characteristic for indicating incidents.

Kim et al. [4] propose a power-aware malware-detection framework that monitors, detects, and analyzes previously unknown energy-depletion threats. The framework is composed of a power monitor which collects power samples and builds a power consumption history from the collected samples, and a data analyzer which generates a power signature from the constructed history. Similarities between power signatures are measured by the $\chi^2$-distance, in order to reduce both false-positive and false-negative detection rates.

Davis et al. [5] propose a host based intrusion detection engine (HIDE). It has the goal to alert a user when a suspected attack is underway before irreparable damage is caused. Therefore, HIDE first monitors anomalous behavior of the battery when it fails

\[^{1}\text{http://www.clamav.net/}\]
to go into suspend or sleep mode. Then HIDE sends an alarm message to the user and the nearest proxy server. It starts to write logs that contain the causes of the higher power consumption. In the next step, HIDE suggests mitigation measures.

Cheng et al. [6] developed a system that uses system and log file monitoring in order to detect malware infections. Therefore, a monitoring client is installed on a Windows Mobile 5 device that is able to determine its own phone number, the date, the cell id, the SMS and calling logs. Statistical and abnormality-based analysis for processing the monitored data is used. For privacy and authentication, ticketing and encryption is introduced. Whenever an infection is detected, the system alerts the corresponding device as well as all devices with contact to the infected one. They evaluate their approach with a simulation based on SMS traces coming from a cellular-network provider from India. The problem about this approach is that malware might circumvent that activities will be logged.

Buennenmeyer et al. [7] present an approach for monitoring current changes on a smartphone in order to detect anomalies. The changes can be caused by malwares and external attackers, e.g. flooding or network probing. The monitored data is sent to a remote server that creates profiles of each monitored device and hence is able to detect anomalies. They evaluate the power consumption of the monitoring and state that the client uses less than 2% of battery resources compared with the corresponding baseline battery lifetime.

Samfat and Molva [8] presented a distributed intrusion detection system for cellular networks that tries to detect abusive behavior like masquerading or eavesdropping. They use learning algorithms in order to obtain user profiles, which in turn are used as signatures to detect abnormal behavior. Example features are call length or cell information. They use network-based intrusion detection and do not try to detect on-device malware. Therefore, this approach is limited to attacks creating network data.

Miettinen et al. [9] designed an intrusion detection framework, which uses host-based and network-based intrusion detection. If an anomaly is detected on a mobile device, the device sends an intrusion alert to a back-end server. This server is able to collect further information from network-based sensors in order to create network related intrusion alarms when necessary. Miettinen et al. do not remotely store or analyze device data. Furthermore, they use a correlation engine in order to correlate the device and network intrusion alarms. The increasing capabilities of mobile devices nowadays allow the deployment of a light-weight version of this approach.

Bose et al. [10] propose behavioral detection framework to detect mobile malware, instead of the signature-based solution currently available for use in mobile devices. They represent malware behaviors based on a key observation that the logical ordering of an applications actions over time often reveals the malicious intent even when each action alone may appear harmless. Also, they propose a two-stage mapping technique that constructs these malicious behavior signatures at run-time from the monitored system events and API calls while studying 25 distinct families of mobile malware in Symbian OS. They discriminate the malicious behavior of malware from the normal behavior of applications by training a classifier based on Support Vector Machines. They offer a good alternative to signature-based detection because of its smaller database that includes behavior signatures for each malware family, and capability of detecting new malware variants.
Schmidt et al. [11] implemented a Symbian OS-based monitoring client that uses a remote server for anomaly detection. The server-side detection framework allows the use of various detection plug-ins. This approach is realizable on most current smartphones where the battery usage of the system depends on the configuration of the communication frequency.

2.2 Intrusion Detection Systems in Ad-Hoc Networks

Traditional intrusion detection systems are not considered sufficient to satisfy the security needs of ad-hoc networks due to their different characteristics from infrastructure networks.

Zhang et al. [12] mentioned that a better intrusion detection in mobile computing environment should be distributed and cooperative. They proposed to use anomaly detection models constructed using information available from the routing protocols for intrusion detection purposes.

The work of Hubaux et al. [13] takes care about selfish node intentionally dropping packets. Here, a stimulation scheme based on a trusted and tamper resistant hardware module was proposed. According to Srinivasan et al. [14], such proposals have been made by several works [13, 15, 16, 17, 18] however that approach has a drawback that involvement of each package results in large overheads.

Huang et al. [19] presented a cluster-based detection approach for intrusion detection system and showed that they could maintain the same level of detection performance as the original per-node detection scheme whereas the host CPU utilization was about 29 percent while the mobility of mobile ad-hoc network was low. Their work had some strong assumption such as the existence of public key technology which is obviously controversial that the maintenance of such a technology may not be available for ad-hoc networks.

Sterne et al. [20] proposed a generalized, cooperative intrusion detection architecture which has a dynamic topology and cluster heads. They pointed the challenges of node mobility and the architecture they proposed was intended to address these challenges.

In fact, all the approaches which are mentioned above have an important value in order to maintain collaboration among distinct nodes in ad-hoc networks. Correlatively, our approach also aims to maintain such a collaboration, but we are not targeting at first the specific challenges arising in ad-hoc networks but to use this networking mode for the scenarios shown in Section 5.3. These scenarios mainly describe cases with existing and absent detection server.

However, the on-going research about security and reliability challenges is interesting for further extensions of our approach.

3 Overall System Architecture

The overall system basically realizes a client-server architecture which can be seen on Figure 1. It basically provides three main functionalities:

1. On-device analysis
2. Collaboration

3. Remote analysis

The client gathers data for on-device, collaboration, or remote analysis. For improving detection, data can be exchanged between two mobile clients. This data can consist, e.g., of detection results or anomalous feature vectors. Whenever on-device detection is not feasible, the client can send data to the remote server. In turn, the server can send detection results back to client. Additionally, it can send commands for reconfiguring the client. For collaboration purpose, the client can exchange monitoring and detection results with other compatible clients.

![Figure 1: Overall System Architecture](image)

4 Monitoring and Detection Client Architecture

Figure 2 shows the architecture of the monitoring and detection client. The bottom-up view on it starts with the Linux operating system level generating signals received by the actual monitoring components. The Linux application level provides all the functionality needed for monitoring and storing device and operating system information. On Java application level anomaly detection, detection collaboration, and detection response are realized where the corresponding states can be visualized in a user interface.

4.1 Linux Operating System Level

The Linux operating system level provides events that are recognized by the monitoring system. These events are initiated by kernel or file system changes.
4.2 Linux Application Level

The monitoring architecture on Linux application layer consists of two programs: the monitoring application and the control daemon. The control daemon is responsible for checking the status and persistence of the monitoring application. The monitoring application extracts information (features) from the Linux kernel and file system. These features are used by the detection for creating a sense of normality. Therefore, the features contain information about the hardware and software states of the device. It has a generic and extensible design for modifying it to corresponding needs.

**Interconnect Daemon** This is the main module of the monitoring application. It is triggered and controlled by the event detection module for generating vectors containing features.

**Event Detection Module (EDM)** This is an essential component of the monitoring system. It recognizes changes in the kernel and file system and generates corresponding events, e.g. new process is started. Basing on these events, features are
extracted that can vary in their content and size. Each feature is marked with a
time stamp and event for later processing.

**Kernel Monitoring Module** This module extracts kernel-based features. Examples
for this are process lists, system call traces, and symbol tables.

**Filesystem Monitoring Module** This module extracts and verifies information on
files. Examples for this are a list of open files or an integrity check on predefined
files.

**Log-file Monitoring Module** Since Android and many applications support logs,
this module extracts information on changes and existence of these.

**Network Monitoring Module** This module can extract information on current net-
work configurations, configuration changes, network status, and network traffic.

**Database Interface (DBI)** This interface provides access to the Android SQLite
database from Linux application level. It is mainly used to store the feature vec-
tors created by the event detection module.

### 4.3 Java Application Level

The monitoring and detection architecture on Java application layer realizes several
tasks for anomaly detection, detection collaboration, and detection response.

**Detection Module** The detection module runs light-weight detection algorithms bas-
ing on feature vector excerpts from the database. It consists of a detection man-
ger co ordinating a variable amount of detection plug-ins. These plug-ins are
instances of detection algorithm that on the one hand can analyze feature vectors
and on the other hand can analyze results from different detection algorithms.
Whenever cooperative detection algorithms are used, it can additionally trigger
the collaboration module.

**Collaboration Module** The collaboration module provides the means to enable de-
tection as well as response in a collaborative manner as an API. Therefore, the
collaboration module stores the node configuration of the device in a dedicated
data model. Based on this model, interests for the collaboration can be defined
that are matched against other node configurations. Thus, partners for the pur-
purpose of collaboration are found and communicated with via the communication
module.

**Response Module** This module enables countermeasures to detected incidents.

**Communication Module** For exchanging feature vectors with the remote server or
collaborative peers, this module provides suitable functions and network access.

**Java Database Interface (JDBI)** This interface provides access to the Android SQLi-
te database from Java application level. It is mainly used to extract feature
vectors and detection results recorded by the system.
Graphical User Interface  This module visualizes current monitoring, detection, collaboration, and response status.

5 General Detection Architecture

An open system like Android requires protection against unwanted software and intrusion. In general there are two principal techniques handling this, namely misuse detection and anomaly detection. The former method is intended to recognize known signatures of malware and attacks, the latter to determine the degree of normality of some observables. Since there is no malware existent for the Android device, our focus is set on anomaly detection. Anomaly detection can be used to identify new unknown attacks, which in turn can be used on- and offline to generate signatures for fast detection in the future. Note that the detection architecture does not need to be changed for misuse detection.

The question arises what normality means. In our approach we distinguish an individual and a common sense of normality. Either are learned statistically and each device can check a system state according to both measures.

When constructing a detection mechanism for a mobile device such as Android, the computational costs has to be kept acceptable due to the limited resources and the need of energy saving. Thus energy efficiency is a guide line for the architecture. Taking this into account, complex computational task and the storage of huge data sets is outsourced to an external server and the on-device detection algorithm is kept relatively simple. Since each detection requires energy, the system integrity should not be checked more often then really necessary, i.e., only on certain occasions. Hence, an event-based approach seems more reasonable than, e.g. a time-periodical one. Furthermore, neighbor devices are taken account in order to collaborate and exchange data in the existence of an ad-hoc network.

5.1 Detection Mechanism

According to our approach five major tasks have to be handled:

1. Event perception, which is done by an event sensor (event detection module (EDM)).

2. System monitoring, to gain informations about some system observables (features) when required. For each class of event there is an adequate monitoring module, recall Figure 2, the entirety of those we will call system monitor.

3. Detection, i.e. analyzing system features and assigning a level of normality, done by the detector, which consists of a detection manager and event-specific detection units and meta detection units.

4. Learning, which the external server is responsible for.
5. Collaboration, which is used in the absence of external server or for reducing the load from the external server.

As the first two units are already described in previous chapters, we present the last three. The detection manager is a daemon, which can be implemented as an Android activity. It is set on auto start and on the highest priority level. The system is prevented from stopping the detection manager via the method `setPersistent()`. In this way, it is assured that it runs permanently in the background. Normally, an activity should not be set persistent since then it blocks system capacities.

The jobs which have to be accomplished by this unit is receiving signals sent by the event detection module and starting a corresponding detection unit. The latter are implemented as sub-activities and assign to each feature vector a level of abnormality and returns it (in a bundle) to the detection manager. If it exceeds a predefined threshold, the detection manager will alert the user via GUI.

The external server does the hard work of statistical learning. The accumulated training data is sent from the database of a mobile device to a server, and in turn the server provides updated parameters for the detection to the mobile device. For a more detailed view on learning see 5.2.

Let the interaction of these units be described by an example. Assume that one of the events we described occurs, say a new process is being launched. This event is sensed by the event detection module, which informs immediately the system monitor and the detection manager about which kind of event has occurred. The system monitor then extracts some (event-specific) system features, in this case the sequence of system
calls caused by this new process along with CPU/memory utilization and other process data. Meanwhile the detection manager has started an event-specific detection unit, i.e. the detection unit corresponding to the “process-started event”. This detection unit evaluates then the level of alert from the feature vector provided by the system monitor.

5.2 Server supported learning

Independent from which reasonable learning technique is chosen, the algorithm for that cannot be performed on the mobile device. Hence, the training data is gradually stored in a database and - after a certain threshold of data amount has been accumulated - sent to a server, where the individual detection parameters are evaluated. Training data is separated according to event class so that event specific detection parameters are determined, which will be sent back afterwards. Each detection unit attains in this way an understanding of normal system behavior which follows each specific event. Furthermore the server also calculates a common sense of normality based on the broad statistical data of all users and makes these common parameters available for the detection units of each user. The reason for that is that user behavior might switch abrupt if, e.g., a new application has been recently installed. Then the detection unit will state a high individual level of alert whereas it will claim that the system behavior is fairly normal relative to other users, since some of which might have already worked with this new application.

5.3 Collaborative Intrusion Detection in Ad-Hoc Network Scenario

In the scope of the proposed detections scheme we consider two scenarios for the collaborative approach: on the one hand, the server might not be available. A dedicated server always represents a single-point-of-failure. In the case of a loss of service or limited access, the collaborative scheme can enable at least a baseline of detection. On the other hand, ongoing-research investigates further collaboration scenarios supplementing the server-based approach by further measures. Here, we consider collaborative file integrity checking as an interesting scheme. In general, the collaborative scenarios can be summarized as following:

- Neighbor does a considerable amount of computation for the initiating device (a)
- Neighbor exchanges data with less computation effort to an initiating device (b)

Keeping in mind those concepts, we have developed two generic protocols which can be used for different events and features. There exists necessity for specifying collaboration partners since possibly not all neighbors have the same interests for collaboration (e.g. One may have a different kernel version which consequently uses different system calls).

Both protocols have a common initial message traffic for the confirmation of the collaborating partners. In order to express a specification, the node configuration is stored on each device. The protocol related to (a) contains message traffic for a specific
computation, e.g. a node sends a feature vector to the neighbors and expects them to compute anomaly status based on their detection unit. Second, the protocol related to (b) contains message exchange of data which considerably requires less effort, e.g. a node requests detectors from his neighbors for a specific event. This example can compensate the absence of a server.

6 Conclusion and Future Work

This work presents an initial architecture for monitoring Android-based devices. Therefore, a dual system is used allowing on-device and remote anomaly detection. Future work will show the feasibility of this approach in terms of resource usage and detection quality.

7 Acknowledgements

We thank Kamer Ali Yüksel, Osman Kiraz, Karsten Raddatz, and Hans-Gunther Schmidt for their enduring work while attending DAI-Labor security course and being internship student.

References


