# Guarding Android – An Implementation Framework for Mobile Sandboxes

*Theodor Nolte*

Projekt 1

*»December 11, 2013«*

*Fakultät Technik und Informatik*
*Department Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

# Contents

# 1 Introduction

Mobile devices have become widely used, mostly as smartphones or tablets. In December 2012 was the first time when the majority of mobile phones sold in the European Union where smartphones [8]. In the worldwide smartphone sales share for 2012, Googles Android dominates with 68,3 %, Apples iOS 18,8 %, and others (inclusive BlackBerry OS and Windows Phone) add to 12,9 % [9].

Because of its enormous deployment mobile devices like smartphones or tablets have become more interesting for attackers. Mainly Android is in the focus of attacks, due to its biggest market share and because of its relatively open Application market (*Google Play* [7]). For the last three month of 2012, the share of newly detected malicious software (malware) families for mobile devices was for Android at 96 % [4]. While in 2010 and 2011 most Android's malware was deployed in the Chinese language area [29], it seems that the malware authors now are acting more in a world-wide scale.

Mobile devices are limited in resources in many aspects. Processors and all other hardware-components are optimized for low power consumption. The operating system and further software of a mobile device must account for the limited availability of resources, too.

Mobile devices largely host private data like contacts, calendar entries, location tracks, private messages, and payment credentials. The more mobile devices are in use for online payments, the more they will come into the focus of attacks, trying to misuse these functionalities to gain illegal profit.

Because mobile devices provide several wireless interfaces and because of their mobility, new strategies on attacking these devices are possible. In general, malware implements functionality the user does not want. This could happen in very different ways and many kinds of malware exist. The authors of malware continuously create new exploitation mechanisms and it is required to have the ability to align the defeating strategies to new attacks.

Oberheide and Jahanian elaborated in [11] that mechanisms of malware-defense for securing traditional computing should not just be taken to mobile environments without justification.

Because the mobile device resources are limited, the performance gap to traditional computing environments will hold on. New mechanisms of malware detection and defense, but also adoptions of traditional mechanisms are needed to provide mobile devices with effective security mechanisms. The challenge here is to not cut down the mobile devices usability.

A framework for malware detection and defense on mobile devices must account for all the aspects mentioned above. It should be secure, flexible, dynamically and statically modifiable, easy to use, and it must use resources carefully – and it must be able to find malicious software, of course.

In this report, we introduce a framework for malware detection and defense on mobile devices. The framework acts in the paradigm of a sandbox in order to protect the mobile device. Our focus is on guarding of mobile devices. This means to achieve a framework which acts dynamically in awareness for current threats against a mobile device.

The remainder of this report is structured as follows. In section 2, the problem of integrating malware detectors will be discussed along with related work. In section 3, background work will be presented. In section 4, a flexible framework for malware detection on mobile devices will be introduced. Finally, this paper concludes in section 5 and gives an outlook.

# 2 Problem Description and Related Work

## 2.1 Integrating Malware Detectors and Defense Mechanisms on Mobile Devices

The domain of malware defense has two big challenges. First to detect and possibly understand malware, and second to defend malware from execution to hinder the evolvement of malicious behavior.

In order to detect and understand malware, information could be achieved by analyzing potentially malicious software and found malware in a dedicated environment where sufficient calculating resources are available. Also detection mechanisms of malicious behavior in productive environments are possible in order to hinder malware infections on such devices.

The detection of malware is not an easy task. It is highly desirable that malicious code will be detected correctly. But it is more important that non-malware will not be recognized as malware (low false positive rate). Even false positive rates bigger than one percent are not acceptable for malware detection mechanisms. It would be fatal if an anti virus software moves an operation system relevant file into quarantine.

Detection mechanisms based on malware signatures are a common method to identify malware. Such a signature could be seen as a feature vector which clearly identifies a special malware after it has been analyzed. But only small changes in the code of the malware will achieve that the signature does not match to the changed malware. Malware signatures can only be created after the identification of malware and must be updated in a regular manner. Therefore, signature based detection mechanisms are not able to detect zero day exploits.

To overcome the limitation not to be able to detect new unknown malware, signature based malware detection mechanisms are often combined with detection mechanisms based on heuristics which try to detect malicious code patterns. Also mechanisms will be used which analyze the behavior of software executed within sandboxes. But both, the heuristic and behavior based methods have a significant lower rate of malware recognition.

Oberheide *et al.* exposed [11] that this *traditional* malware detection mechanisms should not be used on mobile devices without an adoption in respect to the special requirement of mobile devices. The most important requirement here is that a malware detection mechanism must not use much calculation resources.

To hinder malicious behavior on mobile devices, it is common to execute potentially untrusted software within sandboxes.

## 2.2 Related Work

### 2.2.1 Malware Detection and Analysis on Mobile Devices

The analysis of malware is a special kind of analysis of software. In general two kind of analysis of software exist: static and dynamic. Static analysis means to analyze the files of a software while the software is not executed. For a dynamic analysis, the softwares behavior on execu-

tion is observed. To limit the risk of harmful behavior in the latter case the software must be executed in a sandbox.

Due to the limited resources of mobile devices, malware detection approaches often fall back in processing analysis off-device. Bläsing *et al.* [2] present Android Application Sandbox (AASandbox) to statically and dynamically analyze android applications (Apps). Both, the static and the dynamic analysis, will be made off-device, for example on a desktop computer with sufficient resources for calculation. While the dynamic analysis requires more resources than the static analysis, the static analysis also benefits from the higher calculating resources off-line, too. Both analysis are executed fully automatically.

The BMBF research project *MobWorm* [5, 16] presented a forensic tool-chain for malware analysis named Android Data Extractor Lite (ADEL). It extracts data stored in the SQLite databases of an Android device using its Developer Interface. The extracted data will then be parsed and prepared for an analysis report.

Work has also been done in analyzing malware on the mobile device itself. Schmidt *et al.* [13] proposed an approach of static malware detection based on searches for suspicious patterns of Linux function calls. This method is lightweight enough to operate on the mobile device itself.

### 2.2.2 Malware Defense on Mobile Devices

Malware detection can be a part of an *active* malware defense when it is used in order to isolate detected malware and suppress its execution. But more common on mobile devices are *passive* malware defense strategies. This means to hinder potential malicious activities of software by design and has the advantage not to use extra resources. This concept of defense is often called sandboxing.

Egners *et al.* [3] summarize the sandboxing mechanisms used in Android, iOS, and Windows Phone for installed Apps. The sandboxing mechanisms have in common that installed Apps only have the possibility to interact with its environment via defined interfaces.

# 3 Background Work

## 3.1 Entropy-Spectral Analysis

Schmidt *et al.* [15, 14] presented a lightweight method of malware detection based on entropy spectral analysis. The method was elaborated more further in [1].

Novel in this method is the combination of an entropy based statistical analysis with a short-term Fourier transformation (STFT). In this approach, no special signatures of malware are required. This method is implemented in the *EntropyAnalyzer*. The EntropyAnalyzer will be presented in section 4.4.

The analysis results in an evaluation of the existence of binary code for Android devices (elf-arm-32). If the context of the data defines that the data must not be binary, but the analysis found out that the data contains binary data, it is suspected to be malicious. For example, if the data is a downloaded PDF file and the analysis of this file detects binary code, the code is guessed to be malicious.

The method belongs to the data mining domain. In a process named preprocessing feature vectors will be extracted from the raw data. This feature vectors then are the input for a neural network which represents the actual data mining step. Here, the output of the neural network is the classification if a feature vector belongs to the class of feature vectors of binary code or the class of feature vectors of non-binary data.

The topology used for the neural network is a feed forward network. Before the neural network is able to classify feature vectors it must be trained for this task. In a special training method called backpropagation using data samples of binary code and non-binary data the neural network will be specialized for its classification task. While this training needs high calculating time it will be applied off-device on a desktop computer with higher calculating resources instead on a mobile device.

The processing of the analysis can be grouped into four parts. First the data preparation, second and third to apply entropy and STFT calculations on the prepared data (preprocessing) to gain the feature extraction, and fourth the classification of the data based on the calculation results (feature vectors). Each step is described in more detail below. Figure 1 depicts an activity diagram illustrating the entropy analysis process.

**Data preparation** On a sliding window, the raw data will be divided which leads to a sequence of overlapping data windows.

**Entropy calculation** For each of this data windows the Shannon-entropy $H(X)$ will be calculated byte-wise.

**STFT calculation** Then over the entropy values of several windows a short-term Fourier transformation $STFT(m, \omega)$ will be applied.

**Classification** The values from the STFT calculation of several data windows will be inserted as a feature vector into a neural network together with statistical data of the windows

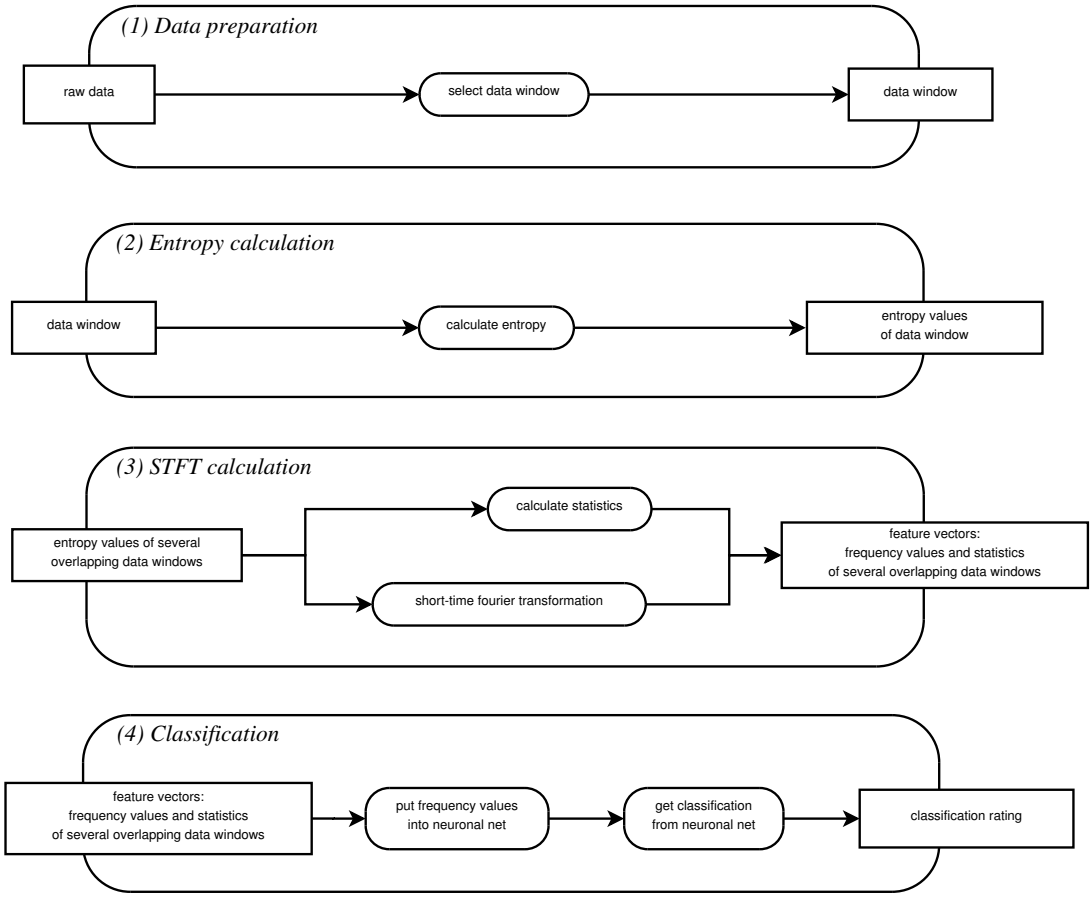



*Figure 1:* Steps of the entropy analysis: (1) Data preparation, (2) entropy calculation, (3) STFT calculation and (4) classification

(arithmetic mean, median, absolute deviation from median). The result of the neural network will be the rating whether the input belongs to the class of executable binaries. Therefore the neural network was trained before with two classes: binary and non-binary data.

These four parts constitute a pipeline, resulting in the result-function. If the result for a window group rates for binary code, the contained data chunk possibly contains binary code.

In future, this method should be optimized in order to lower the false positive rate (that non-binary data will be determined as binary code). Further, the neural network could be extended in order to be able to classify feature vectors into several classes of data types (for example text files, xml/html, binary data).

## 3.2 Appsicht: An App Verification and Process Authentication Architecture on Android

Ugus *et al.* [24] presented a security architecture for the measurement of the runtime integrity of processes and Apps on Linux-based mobile devices called Appsicht. The implementation runs on Android. It uses a whitelist to define the Apps which are allowed to be executed on the mobile device. For every allowed App, a hash of its code will be stored in the whitelist. The execution of other Apps or any whitelisted App which code has been altered, will be prohibited. For trustworthiness, the hashing mechanism, the whitelist itself, and the processes which realize this security architecture are secured against any modification. In order to achieve this, Appsicht is based on a Mobile Trusted Module (MTM) as a trust-anchor [25].

The MTM is a dedicated chip providing a software interface with security related functionalities like the calculation of SHA-1 hashes, the creation of RSA-2048 certificates, generation of random numbers, and secure storage of Critical Security Parameters (CSPs) for example private keys or hashed measurements of system states. This functionalities cannot be manipulated by software. Also physical manipulation of the chip or the extraction of CSPs is hindered by hardware design. The applied MTM standard version 1.0 [21] is the mobile version of the Trusted Platform Module (TPM) standard version 1.2 [20] that shares parts of its interface definition. [19] Both standards are defined by the Trusted Computing Group (TCG) [23] an industrial non-profit standardization organization.

Appsicht uses the MTM functionalities for a whitelisting of dedicated versions of Apps. On Android, an App is a single, zipped file (*.dex). On setup, a hash value will be calculated for every allowed App by the MTM and inserted into the whitelist. Just before an App will be executed, its hash will be computed again and the result will be validated against the corresponding value in the whitelist. Only if they match, the App will be started.

This authorization and validation mechanism is achieved by extending the Linux based Android operating system with new and modified kernel modules. Also the Zygote process, the process which starts all Apps, is modified. All the functionalities of Appsicht must be trustwor-

thy. It will be achieved in extending an assumed secure boot process in validating the integrity of all components of Appsicht.

The intended use of Appsicht is to control that on enterprise smartphones only a predefined set of Apps could be run which has been defined before by an administrator of the company. Appsicht also can be used by the framework presented in this report in order to ensure the integrity of its components (*cf.* chapter 5).

The authors of [24] mentioned that no MTM chip was available for mobile devices. On an OMAP4430 Panda-Board [12] running with Android 2.3.3 (Kernel 2.6.35) they used a software implementation emulating the MTM instead.

TPM 2.0. [22] the successor of the TPM 1.2 standard is a complete redesign focused on flexibility. It is suited not only for business environments on desktop computers but also for various kinds of consumer devices. This standard seems to be more promising for success: Every tablet which complies with the *Windows 8 Hardware Certification Requirements* [10] must be equipped with a TPM 2.0 chip. Future implementations of Appsicht should be adoptable for using TPM 2.0 instead of MTM 1.0.

# 4 A Flexible Framework of a Mobile Sandbox

On mobile devices, new detection and defense strategies are needed. Common methods of malware detection used on desktop computers like virus scans based on malware signatures consume too much of resources. A big challenge in general and also on mobile devices is to detect unknown malware (zero day exploits).

## 4.1 Objectives

The goal of this work is to integrate versatile methods of malware analysis, detection, and defense on mobile devices. The methods are developed within the SKIMS-Project [17] and focus on novel approaches. For example, the Entropy-Spectral Analysis (*cf.* chapter 3.1), requires its integration in the wider context.

Therefore, we want to create a framework which is able to integrate threat detectors with defense components on a mobile device. The framework must be extensible with respect to the integration of further mechanisms for analysis, detection, and defense, and has to be build up of modules. Further, it must be adaptable to future requirement changes as it is intended for the implementation on different mobile platforms and architectural changes of new platform versions. Due to the restricted resources on mobile devices, the framework must use the detection and defense mechanisms in a lightweight manner. The framework should interfere with the user as little as possible. Also, the framework must use as little resources as possible.

In order to defend the mobile system against a current threat, the execution of malicious software must be avoided. This means to disable or change functionalities of the mobile system and to prevent malicious software to be executed and suspicious data to be used.

Detection and defense must interact with each other. This could be realized by mutually interacting components. But a central controller which manages the events from the sensors and analyzers and which triggers the defense mechanisms has several advantages. The fact that the components only communicate via a controller and not directly at each other, correlates to loose coupling. This conduces the extension with new components for detection and defense. Also it is easier to keep track of all.

As mentioned in 2.2.1, the analysis of malicious software could be achieved statically and dynamically. To recognize malicious behavior of executed software and in the interaction with other systems we need appropriately placed sensors.

For example, a sensor could be a receiver listening for warning messages from other devices. Or a honeypot on a mobile device listening on unused ports could act as a sensor. When a file was caught by the honeypot, an analyzer, which is not a sensor, could then search whether malicious data is hidden in the file. Both, sensors and the analyzers are able to detect threats.

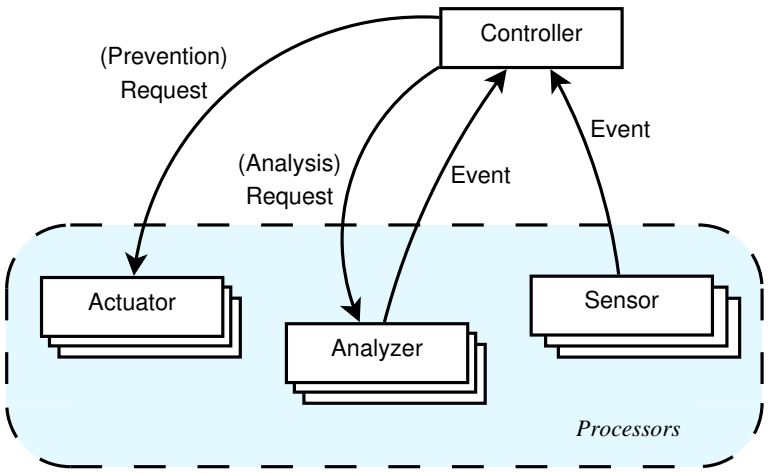



*Figure 2:* Logical context of the framework

## 4.2 Concept of Core Components

Figure 2 shows the logical context of the framework with one controller and several processors. The following roles exist:

**Controller** Based on the incoming events the controller evaluates the current threat level and effects protective actions in sending appropriate requests to actuators.

**Sensor** A Sensor detects security related events and sends them to the controller. For example, the Honeypot *Droidspot* in the SKIMS demo setup takes this role.

**Analyzer** Triggered by requests coming from the controller the analyzer starts analyses. Results of such analyses will be send as events to the controller. The EntropyAnalyzer is an example for an analyzer.

**Actuator** An actuator is a defense component. It can enable and disable security related configurations of the mobile system, for example switching the Wifi connection on or off.

A processor could be an actuator, analyzer, sensor or two or three of them at the same time. All these components are loosely coupled since the clients will communicate only with the controller using well defined interfaces. They do not communicate among each other.

Based on incoming events, a controller evaluates the current threat level and reacts by sending appropriate messages to the mobile devices actuators.

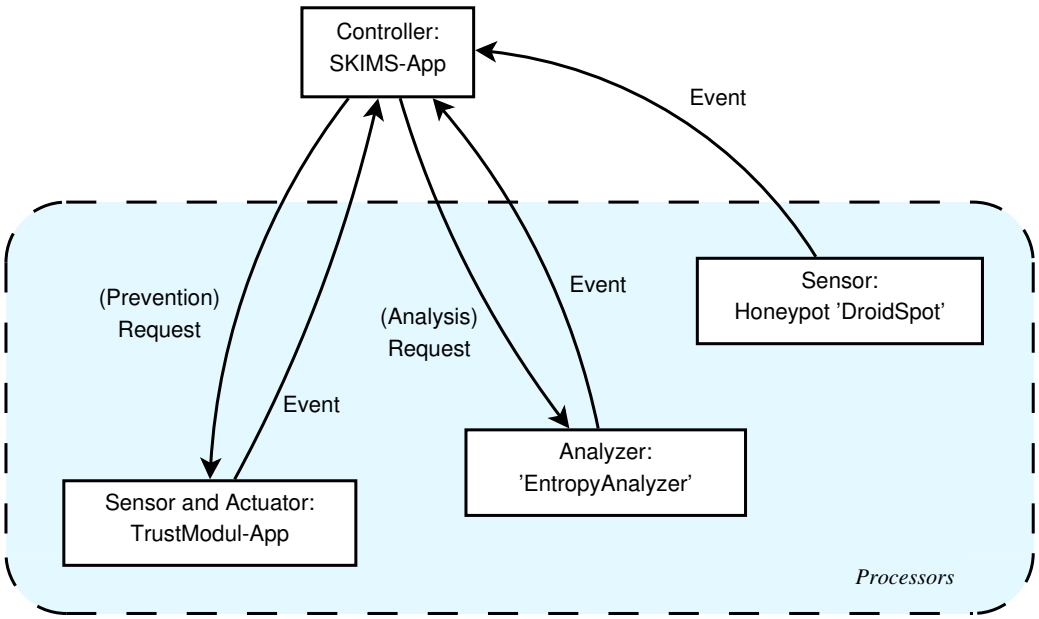



*Figure 3:* Business context diagram of the implementation framework

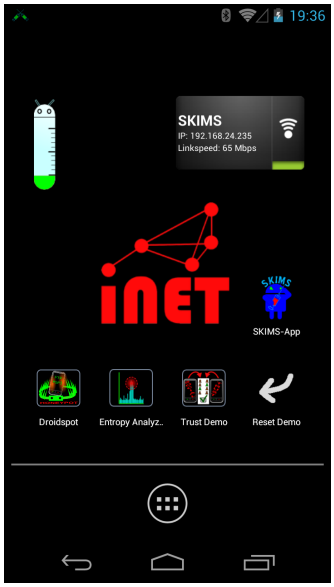
## 4.3 Implementation

### 4.3.1 Components of the Framework

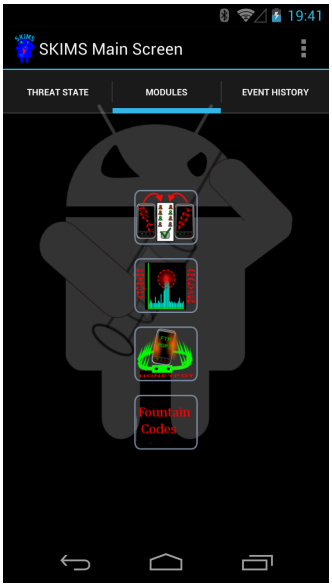
The business context of the framework is shown in figure 3. It is implemented for mobile devices with Android and consists of several components which are realized as Android-applications (Apps):

**SKIMS-App** The SKIMS-App is the controller of the framework. It signalizes the current threat level to the mobile devices user by a symbolic traffic light with the states green (no threat), yellow (threat possible) and red (threat detected). Figure 4 shows several views of the SKIMS-App. Its business logic and interaction with the other components is described in subsection 4.3.2.

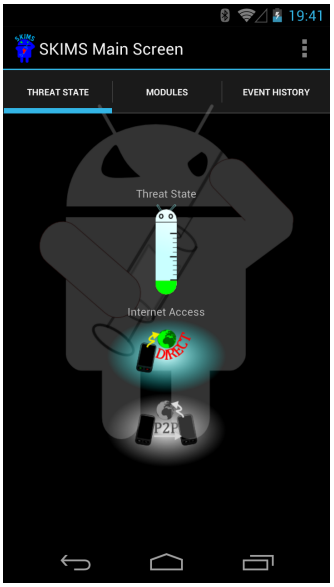
**DroidSpot** DroidSpot [26, 28], a honeypot App, is a sensor of the framework. Beside of faking an email-server in spoofing SMTP, IMAP and POP3 services, and also spoofing an ssh service, it listens for FTP connection attempts. On an Android device in production mode no FTP service is active. So, every access via FTP can be considered as an attack. An attacker can start an anonymous-session with a faked directory-list and is empowered to upload a file. At the begin of the FTP-session, DroidSpot informs the SKIMS-App about the misuse of the mobile device. Also, if a file was uploaded, DroidSpot sends an appropriate message to the SKIMS-App. Figure 5 shows the view of Droidspot.




(a) Home screen with SKIMS-App thermometer widget in the upper left area




(b) Modules of the SKIMS-App




(c) Current threat state and used internet connectivity

*Figure 4:* Several views of the SKIMS-App


*Figure 5:* View of Droidspot

| **Row Name** | Java Data Type | *Meaning* |
|---|---|---|
| **id** | int | *Primary key* |
| **moduleid** | int | *Id of module (processor) which originated the event (1: Droidspot, 2: EntropyAnalyzer, 3: Trust-Module)* |
| **timestamp** | long | *When the event was created* |
| **category** | String | *General category of the event* |
| **value** | int | *Threat level of the event (evaluated by processor)* |
| **infotext** | String | *Information for user presentation* |
| **data** | String | *Additional context sensitive information (optional)* |

*Table 1:* Database scheme of table threatdb of the controller (SKIMS-App)

**EntropyAnalyzer** The EntropyAnalyzer is an analyzer component and is presented in detail in subsection 4.4.

**Trust-Module** The Trust-Module App [18] is a sensor and also an actuator. Two mobile devices where this App is installed on, are able do determine whether they can trust each other. It is a transient kind of trust. If they have the same member in its contacts and if both trust this member, than they trust each other. Now, if one mobile device registers an attack originated from its Internet connection, it could initiate a handover to use the Internet connectivity of the trusted neighbor. Figure 6 shows several views of the Trust-Module.

The components were developed and implemented by several groups. In meetings we developed and discussed the interface definition of the interaction of the components.

### 4.3.2 Controller: Control Flow and Data Exchange of the SKIMS-App

Every event sent from a processor will be received by the SKIMS-App which stores all events in a sqlite database table named *threatdb*. The *threatdb* database scheme is shown in Table 1. As of the id which is created by the SKIMS-App, all data of an entry comes from a received event.

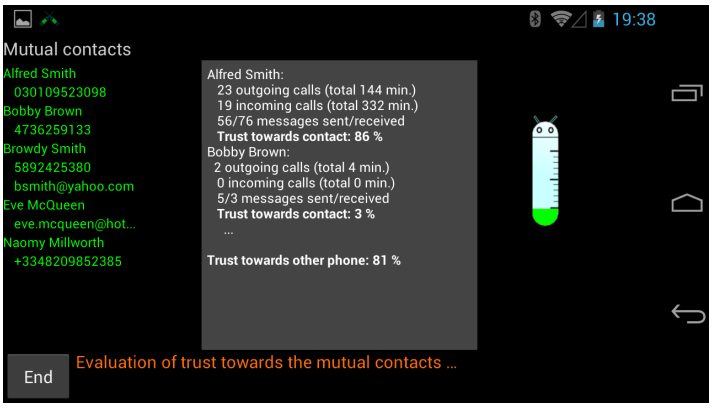
The incoming interface of the SKIMS-App utilizes the Android framework and is realized by a ContentProvider class of the Android framework. With this class the method `insert()` will be provided with the following signature:

```
public synchronized Uri insert(Uri uri, ContentValues values);
```

The parameter `values` of type `ContentValues` represents a dictionary of key-value-pairs. A processors event here is a `ContentValues` object which has set the keys `moduleid`, `timestamp`, `category`, `value`, and `infotext`. Optionally a value for the key `data` could be set, but not every event needs additional data. The data types of the values are listed in the second column of Table 1.


(a) Double encryption of local and remote contacts


(b) Comparison of double encrypted contacts for mutuals




(c) Trust evaluation of mutual contacts

*Figure 6:* Several views of the Trust-Module App

```
1  ContentValues vals = new ContentValues();
2  vals.put("moduleid", 1);
3  vals.put("timestamp", System.currentTimeMillis());
4  vals.put("category", "FTP␣Login␣Attempt");
5  vals.put("value", 2);
6  vals.put("infotext",
7          "Attempted␣login␣to␣FTP␣honeypot␣-␣root;letmein");
8  ContentResolver cres = getContentResolver();
9  Uri eventdbUri =
10         new Uri(content://com.escrypt.skims.threatdbprovider/events);
11 cres.insert(eventdbUri, vals);
```

*Listing 1:* Creation of a Droidspot event

In Listing 1 the creation of an event and its forwarding in Droidspot is shown. Other events originated by this module and by other modules will be created in the same manner. After creation, Droidspot 'sends' the event to the SKIMS-App by storing the event in the database table *threatdb*. For this task a ContentResolver instance will be used (as a counterpart of the ContentProvider).

Figure 7 shows the control flow of the controller (SKIMS-App). If in the controllers database an event originated by a processor will be stored, the business logic unit of the controller will be informed about the event *(step 1 and 2 in the Figure)*. For example, the Droidspot App sends a FTP-upload-event to the controller *SKIMS-App*. This event will be stored into the *threatdb* and the controllers business logic unit will be triggered. Then, the business logic sends an order request to the analyzer processor *EntropyAnalyzer (step 3)*. This order request will be stored into the threatdb, too. So the SKIMS-App can keep track of everything *(step 3b)*. When the analyzer processes an analysis from the request it sends events belonging to this analysis to the controller, for example the start of the analysis or its result *(step 4)*. This again triggers the controllers business logic *(step 2)*.

Table 2 lists all categories and values used in event messages by the modules (processors) of the SKIMS demo. In a first draft, the *category* defines to which class the event belongs to. The value stands for the threat level of this event. Droidspot follows this semantic and sends events following this scheme, but while the value every time is 2 it becomes meaningless.

We have worked out, that an evaluation of the threat level should only be made by the controller and not by the processors. Because only the controller is aware of all events from all processors. This leads to another semantic. *category* stands for the general category of an event, *value* represents the class of the event within that category. The EntropyAnalyzer and the Trust-Module follow this semantic.

As the processors are stateless in the sense of the current threat level, another mechanism for the request messages is used instead of the pair of ContentProvider and ContentResolver.

| **category** | *meaning* | **value** | *meaning* |
|---|---|---|---|
| **Droidspot (moduleid: 1)** | | | |
| **0** | *Unknown* | **2** | |
| **1** | *FTP Login Attempt* | **2** | |
| **2** | *FTP Anonymous Login* | **2** | |
| **3** | *FTP Download Attempt* | **2** | |
| **4** | *FTP Upload Attempt* | **2** | |
| **5** | *FTP Active Bounce* | **2** | |
| **6** | *Malware Upload* | **2** | |
| **7** | *SMTP Mail Attempt* | **2** | |
| **8** | *POP3 Login Attempt* | **2** | |
| **9** | *IMAP Login Attempt* | **2** | |
| **10** | *SSH Login Attempt* | **2** | |
| **EntropyAnalyzer (moduleid: 2)** | | | |
| **1** | *entropy analyzer event* | **1** | *no malware found* |
| | | **2** | *analysis started* |
| | | **3** | *malware found* |
| | | **4** | *analysis finished* |
| | | **5** | *analysis aborted* |
| **Trust-Module (moduleid: 3)** | | | |
| **1** | *connection (attempt)* | **1** | *failed* |
| | | **2** | *connected, but low trust in peer* |
| | | **3** | *success* |
| **2** | *peer discovery* | **1** | *no peer found* |
| | | **3** | *one peer found* |

*Table 2:* Categories and data values of the events sent by the modules

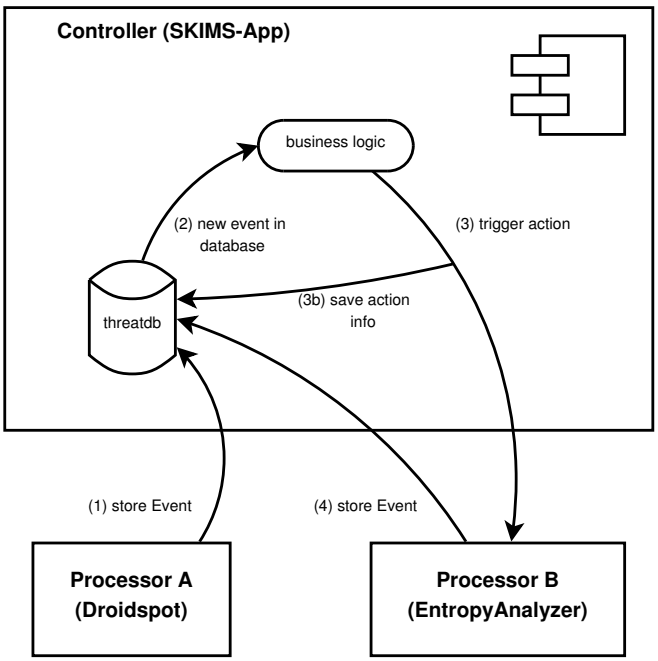



*Figure 7:* Flow control of the controller

Also the Android framework will be utilized. Every processor which is able to receive requests from the controller has a *BroadcastReceiver*. If – for example – the SKIMS-App sends an order request to the EntropyAnalyzer, the SKIMS-App sends an (broadcast) *Intent* to the BroadcastReceiver of the EntropyAnalyzer.

The broadcast messages to the different modules of the SKIMS demo framework does not have a common structure yet (as the events have it). The values used in broadcast messages sent to the EntropyAnalyzer are shown in Table 3.

Both mechanisms of the Android framework, for storing values in a sqlite database and sending an intent message via broadcast, use inter-process communication (IPC) calls which

| **action** | **uri** | **start analysis** | *intention* |
|---|---|---|---|
| | **– (not set)** | **–** | *Switch to EntropyAnalyzer* |
| **RESUME_ENTROPY_ANALYZER** | **file:///path/to/file** | **–** | *.. and open an analysis* |
| | **http:///url/to/file** | **true** | *.. and start an analysis* |
| **START_BACKGROUND_ANALYSIS** | **file://path/to/file** | **–** | *Start new analysis in background* |

*Table 3:* Values of Intent messages sent by the controller SKIMS-App to control the processor EntropyAnalyzer

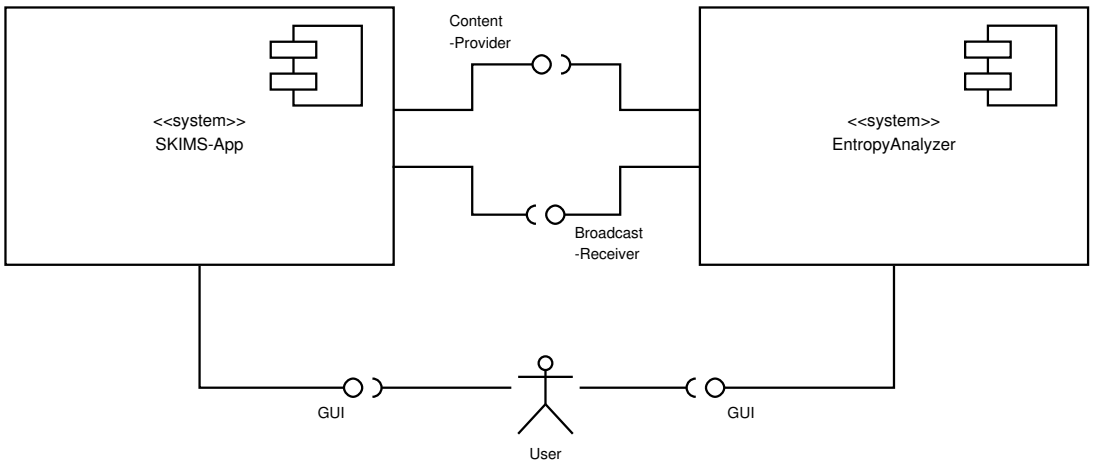



*Figure 8:* Technical context diagram of the EntropyAnalyzer App

are completely capsuled by the Android API [6].

## 4.4 Usecase: EntropyAnalyzer

The *EntropyAnalyzer* is a Java-application for the Android framework (*App*). It was constructed by the author as a demo-application showing the entropy analysis on a smartphone.

Its general functionality is to select a file locally or via http from any server and to analyze the file while downloading it. In real time the result of the analysis will be plotted. Also current status messages (for example *"binary code detected"*) will be send by an Intent to the Controller-App, the *SKIMS-App*.

The analysis was already described in detail in 3.1. On a sliding window over the data stream of the file the EntropyAnalyzer rates for binary code (elf-arm-32) or non-binary data. If at least one chunk was recognized as binary code the raw data possibly contains binary code.

**Implementation of the EntropyAnalyzer** Figure 8 shows the technical context of the EntropyAnalyzer App. Apart from user interaction, the EntropyAnalyzer only interacts with the SKIMS-App.

The EntropyAnalyzer supplies an asynchronous (incoming) interface which is realized by a BroadcastReceiver class of the Android-API. This allows the SKIMS-App to start an analysis. The BroadcastReceiver receives Intent-Objects (requests) from the SKIMS-App. Such an request contains a URI addressing the data to be analyzed. The analysis then will start in foreground or in background. It is also possible to bring the EntropyAnalyzer into foreground showing a running or already finished analysis which was started before as a background analysis.

The EntropyAnalyzer informs the SKIMS-App by Events into a shared database provided by the SKIMS-App using a ContentProvider instance. Life-cycle events inform the SKIMS-App if an analysis had been started, finished, or aborted. Analysis-related events report if the analyzed data was determined as binary or non-binary. In order to allocate an event to an analysis-order originated from the SKIMS-App the each event contains the URI of the data being analyzed.

Figure 9 shows the classdiagram of the EntropyAnalyzer App. One part of the EntropyAnalyzer - the App-part - is for the interaction with the user and with the SKIMS-App. Only there, the Android-API will be used. All other functionalities are placed in the part 'Analysis' and the part 'Entropy-Analysis'. Both do not use the Android-framework. The goal of this separation is to keep the core implementation platform-agnostic and make the implementation portable to other mobile platforms.

The AnalysisController singleton-object manages the analyses, following a model-view-controller pattern. The property class Analysis holds the meta information like an internal used analysis-id, the current state of the analysis and the (intermediate) results. The states are: initial, running, and done. This class is loosely coupled with the analysis process which happens in the class EntropyAnalyzer. The analysis process runs in an own background-thread. Results of type EntropyAnalyzerStatusMessage reach the class Analysis utilizing the Observer pattern. So the way back from the class EntropyAnalyzer to the class Analysis is loosely coupled, too. The GUI shows a graph of the analysis progress (cf. figure 12) and needs the intermediate results. Therefore, the ObserverService class from the GUI observes with the interface IntermediateResultsObserver the Analysis which uses the class AnalysisObservable. The SKIMS-App in contrast only needs to know if the state of an analysis has changed. This will be achieved by the class OtherAppNotifierService which observes with the interface LifeCycleObserver the class Analysis. If a new life cycle event occurs it notifies the SKIMS-App in writing an event into the event database using the BroadcastReceiver.

Figure 10 shows the steps of an analysis which was initiated by the SKIMS-App. The Intent coming from the SKIMS-App contains the URI about the data which has to be analyzed. This URI is passed to the AnalysisController which opens the data belonging to the URI as a data stream and creates a new analysis with this data stream.

**Usage of the EntropyAnalyzer** The EntropyAnalyzer provides an user interface with an intuitive design:

- On start there is the face plan (view of the StartActivity, Figure 11(a)). In the upper left is a text field showing the currently selected source-URI. Because we haven't selected any source this field is still empty.

  On the right-hand side to the text view is placed an image showing the SKIMS-thermometer. It visualizes the current threat level: A green thermometer stands for 'everything is fine' because no threat was detected. Either by result or because nothing was analyzed (initial threat level). A yellow thermometer means that the current threat level is unknown.

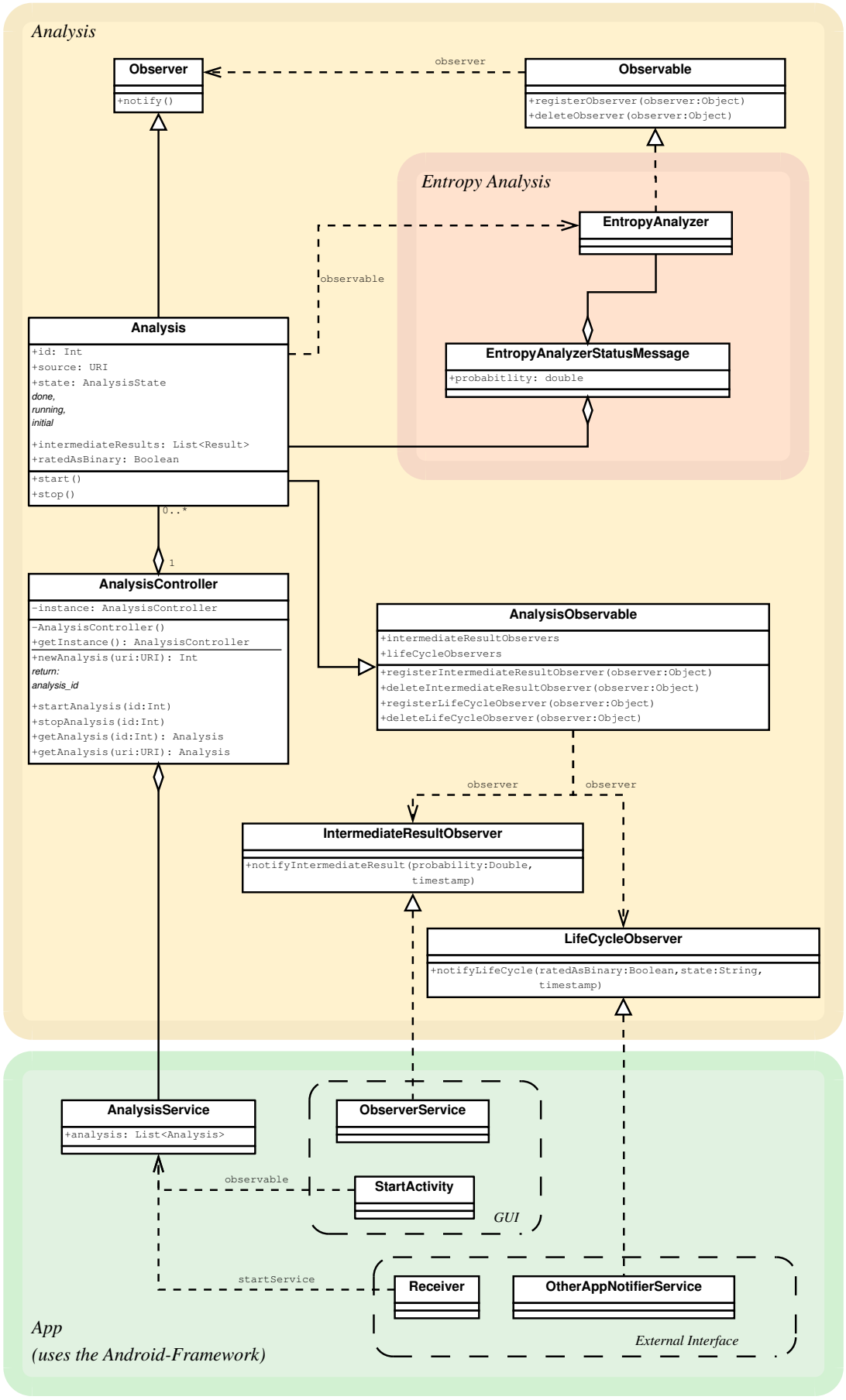



*Figure 9:* Classdiagram of the EntropyAnalyzer App

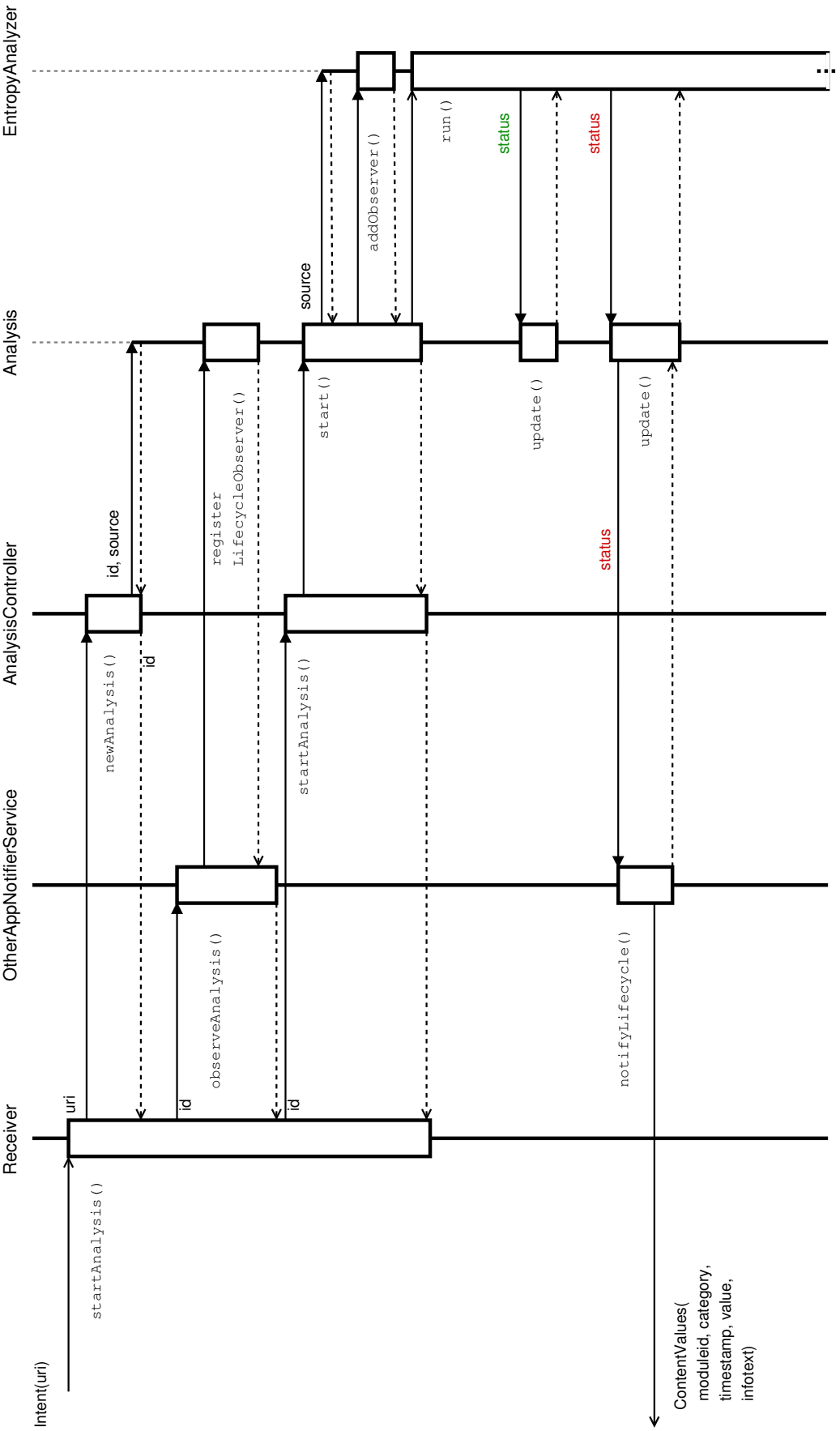



*Figure 10:* Sequence diagram showing the interactions of an analysis running in background

This state will appear if an analysis had been started but no result was received (till now). The thermometer will switch to red if some result of the analysis was greater than 0.5.

On the center is a (currently empty) pane for a plot showing the analysis results – the probabilities of binary data – over the byte stream. The graph will be plotted in a healthful green if the data was recognized as non-binary. If the result is bigger than 0.5 the sliding window was recognized to be binary code and the graph will be filled with an alarming red.

On the bottom are two buttons. The upper and bigger one is for the selection of a source, to start and to stop an analysis (texts 'Select Source', 'Start Analyzer' and 'Stop Analyzer'). The other resets the App.

- The first action is to select the source we want to analyze. (So we have to push the button 'Select Source'.) Three ways of selection are possible (Figure 11(b)): First is the ability to scan a QR Code (Quick Response Code) in which an URI was encoded. This is the main use case (Figure 11(c)). Further it's possible to select a local file (via an external file browser App) or to type the URI by hand.

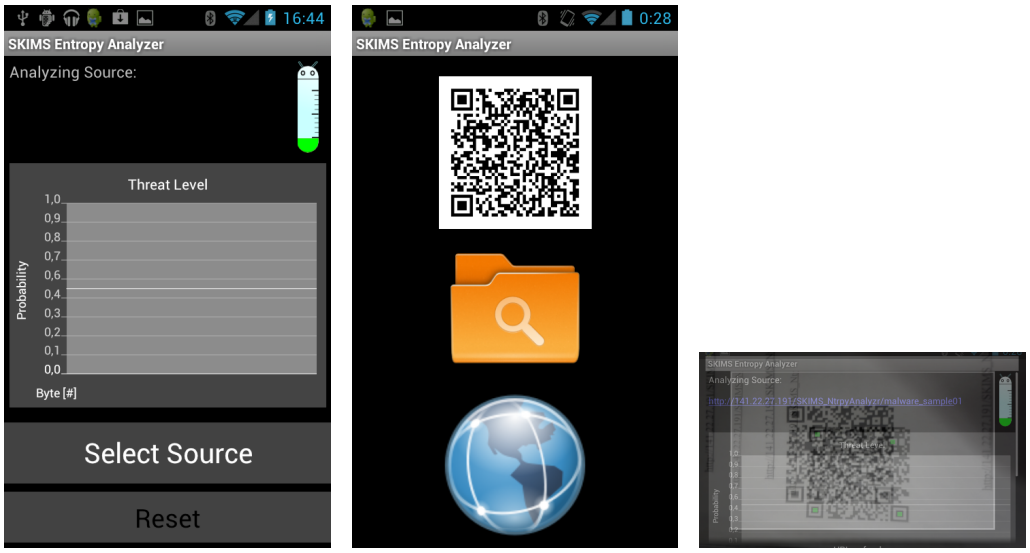
- After the source was determined we are back to the face plan. Now we can see the selected URI. If the URI is an http-URL it is highlighted as a link. Pushing on it would open the link by the web-browser-App of the Android device.

- After pushing 'Start Analyzer' the EntropyAnalyzer will download the file via http or open the local data stream. While loading the entropy-analysis will be running in background. New results will be plotted (the thermometer will change it's color accordantly) and send as status messages to the SKIMS-App (Figures 12(a), 12(b)).

## 4.5 Evaluation of Results

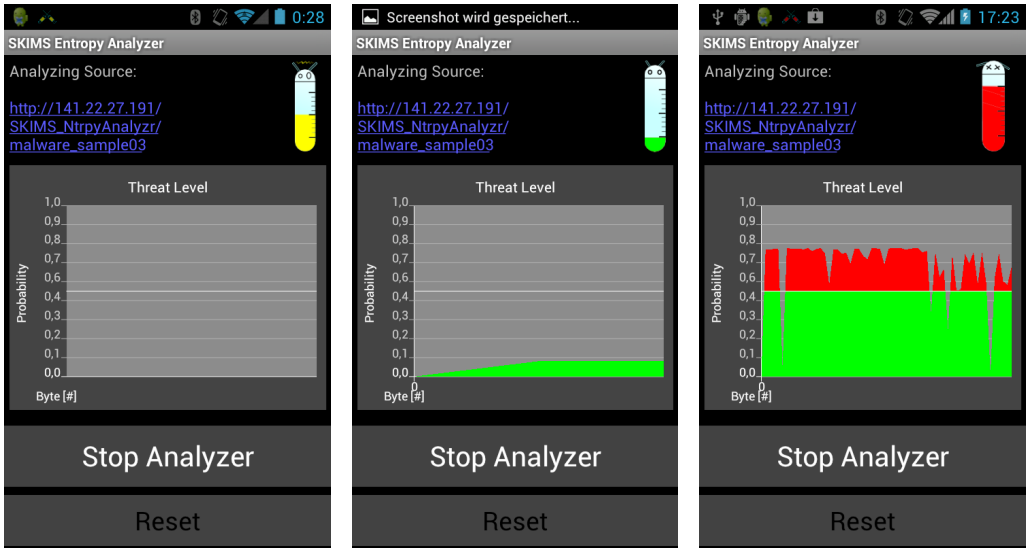
**Demonstrator** The SKIMS-App, the EntropyAnalyzer and the other Apps presented in this section are prototypes with a focus to work as a demonstrator. They show as a proof-of-concept that the framework can be implemented. More further, it was reached a good usability. Even the calculation time intensive analysis of the EntropyAnalyzer performs well. The components of the prototype all use the Android framework and integrate well into the Android user environment. But the complexity of integration was kept low due to the consequent implementation of the components as modules. The modules communicate only using interfaces based on the Android framework and are easy to expand or to exchange.

The demonstrator was presented at the CEBIT 2012 in Hannover and at the ACM SIGCOMM 2012 in Helsinki during a demo session [27]. Figure 13 shows a picture taken from the demo session.




(a) Initial face plan of the Entropy-Analyzer (b) View of the activity SourceSelection (c) Selection of a malware sample via QR Code

*Figure 11:* Several views of the EntropyAnalyzer




(a) Starting an analysis (b) Running an analysis - no binary code detected (c) Running an analysis - binary code detected

*Figure 12:* Progress of an analysis

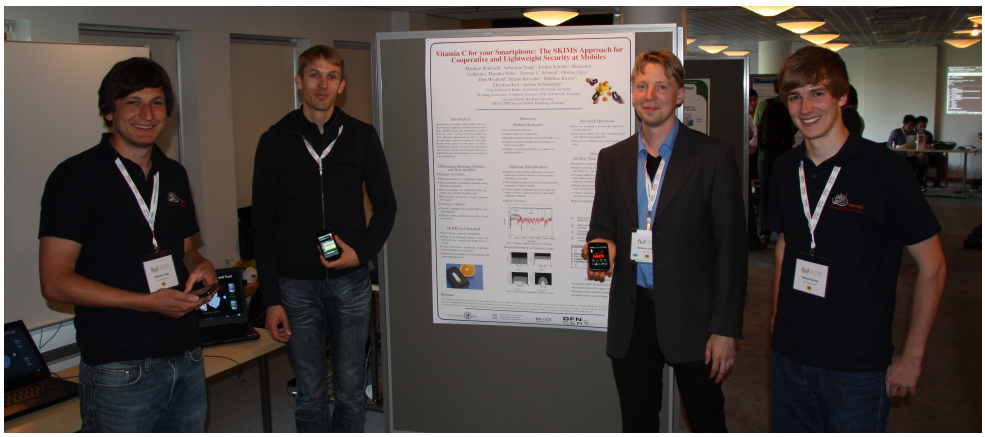



*Figure 13:* Presentation of the SKIMS Approach at the ACM SIGCOMM 2012 in Helsinki

In the presentation the interaction of the frameworks components was shown. At first, a file was uploaded to Droidspot using a graphical FTP client. Then, Droidspot notifies the SKIMS-App which orders the EntropyAnalyzer for an analysis of the file. If the EntropyAnalyzer reports the SKIMS-App that the file contains binary data, the SKIMS-App orders the Trust-Module for initiating a handover of the Internet connectivity.

**Extensibility** The implemented methods of malware analysis, detection, and defense have been developed over time. The client-modules were not available at the same time and its interfaces have been changed during its development. It has been shown that the framework was able to integrate all this different methods of malware detection and defense and that it is extendible for new components. Also requirement changes could be achieved in extending the interfaces parameters.

If a new method of malware detection arises, a module which implements this new feature could be integrated into the framework. Using the same technology new methods of defense could be adapted, too.

**Complexity of Integration** The modularity of the frameworks is shown by the fact that the controller and clients of the framework are realized as Apps. They are loosely coupled and interchangeable with alternative implementation as they communicate using interfaces only. The complexity of integration of the framework at all is not significant higher as the sum of the complexities of its modules.

**Open Aspects** The SKIMS-App provides a single interface for all clients to be informed on new (status-) events. Such an event contains a dictionary (key-value pairs) with the keys *moduleid, timestamp, category, value, infotext, data*. The *moduleid* identifies a client. *category*,

| **category** | *meaning* | **value** | *meaning* |
|---|---|---|---|
| **Droidspot (moduleid: 1)** | | | |
| **1** | *FTP* | **1** | *FTP Login Attempt* |
| | | **2** | *FTP Anonymous Login* |
| | | **3** | *FTP Download Attempt* |
| | | **4** | *FTP Upload Attempt* |
| | | **5** | *FTP Active Bounce* |
| | | **6** | *FTP Data Upload* |
| **2** | *SMTP* | **1** | *SMTP Mail Attempt* |
| **3** | *POP3* | **1** | *POP3 Login Attempt* |
| **4** | *IMAP* | **1** | *IMAP Login Attempt* |
| **5** | *SSH* | **1** | *SSH Login Attempt* |
| **EntropyAnalyzer (moduleid: 2)** | | | |
| **6** | *entropy analyzer event* | **1** | *no malware found* |
| | | **2** | *analysis started* |
| | | **3** | *malware found* |
| | | **4** | *analysis finished* |
| | | **5** | *analysis aborted* |
| **Trust-Module (moduleid: 3)** | | | |
| **7** | *connection (attempt)* | **1** | *failed* |
| | | **2** | *connected, but low trust in peer* |
| | | **3** | *success* |
| **8** | *peer discovery* | **1** | *no peer found* |
| | | **3** | *one peer found* |

*Table 4:* Categories and data values of the events sent by the modules (optimized version of Table 2)

*value*, *infotext*, and *data* differ between different clients in its values and in its semantic, too (cf. Table 2). In the other way – from the controller to a client – for every client an individual interface will be used. This means that for every new client the controller must be adjusted to its specific in- and outgoing interfaces. In order to become a framework where new clients easily can be added, a more generic in- and outgoing interface is desirable.

Table 4 shows an optimized version of Table 2. The *category* identifier is not unique within one module anymore, but unique over all modules. The former categories of the Droidspot are moved to *values* and are subdivided into (real) categories. This changes make the structure of all events consistent and provide a step to achieve a more generic interface.

# 5 Summary and Outlook

Several approaches for malware detection on android focus on the analysis off-device or in a forensic manner. This paper introduced a framework which focus is to guard a mobile device while it is in productive usage. A flexible framework was presented with one controller placed in the middle and several clients around. Novel lightweight detection and defense strategies are implemented in a prototype as a proof-of-concept.

Due to the fact that the implementation of the framework was achieved as a prove of concept and for demonstration purposes, some of the adopted mechanisms are too simple for a use in productive mobile environments:

- The messages of the communication between the controller (SKIMS-App) and the clients (DroidSpot, EntropyAnalyzer, TrustModul) should be secured in using encryption.
- The Apps are developed by different developers. In order to be able to communicate to each other we discussed and agreed to interface definitions. But until now they are individual for each client. It is desirable to extract a common interface for all clients.
- A common interface would allow to extend the controller for an automatic registering and unregistering of clients. Then, for a new client only a new App needs to be installed. Challenge for this feature is to ensure that only authorized clients can be added to the framework.

Notably for the last point the integrity of the controller is important. It could be achieved in using Appsicht (*cf.* chapter 3.2) to guarantee that an unmodified controller is executed on a mobile device.

The entropy-spectral analysis depends on the context of the analyzed data. Because of the fact that the context of traffic data in browsers is clear, it should be promising to implement the entropy based analysis as a browser plugin, for example in Mozilla Firefox and Google Chrome for Android.

## References

[1] Benjamin Jochheim. On the Automatic Detection of Embedded Malicious Binary Code using Signal Processing Techniques – Project Report. http://inet.cpt.haw-hamburg.de/teaching/ss-2012/master-projects/benjamin_jochheim_pr1.pdf, October 2012.

[2] Thomas Bläsing, Aubrey-Derrick Schmidt, Leonid Batyuk, Seyit A. Camtepe, and Sahin Albayrak. An android application sandbox system for suspicious software detection. In *5th International Conference on Malicious and Unwanted Software (Malware 2010)*, Nancy, France, 2010.

[3] André Egners, Björn Marschollek, and Ulrike Meyer. Hackers in Your Pocket: A Survey of Smartphone Security Across Platforms. In *RWTH Aachen University, Technical Report, AIB-2012-07*, Aachen, DE, May 2012.

[4] F-Secure. Mobile Threat Report Q4 2012. http://www.f-secure.com/static/doc/labs_global/Research/Mobile%20Threat%20Report%20Q4%202012.pdf, 7. March 2013.

[5] Felix Freiling, Sven Schmitt, and Michael Spreitzenbarth. Forensic Analysis of Smartphones: The Android Data Extractor Lite (ADEL). In *The 2011 ADFSL Conference on Digital Forensics, Security and Law*, ADFSL 2011, Richmond, Virginia, USA, May 2011.

[6] Google Inc. Content Provider Basics - Android Developers. http://developer.android.com/guide/topics/providers/content-provider-basics.html, 2. December 2013.

[7] Google Inc. Google Play. https://play.google.com/, 5. August 2013.

[8] heise mobil. Marktforscher: Jedes zweite Handy in der EU ein Smartphone. http://heise.de/-1833455, 2. April 2013.

[9] International Data Corporation (IDC). IDC Worldwide Mobile Phone Tracker. http://www.idc.com/getdoc.jsp?containerId=prUS23818212#.UWpwXKovBQK, 4. December 2012.

[10] Microsoft Corporation. Hardware Certification Requirements for Windows 8.0. http://download.microsoft.com/download/A/D/F/ADF5BEDE-C0FB-4CC0-A3E1-B38093F50BA1/windows8-hardware-cert-requirements-system.pdf, 18. September 2012.

[11] Jon Oberheide and Farnam Jahanian. When mobile is harder than fixed (and vice versa): demystifying security challenges in mobile environments. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, HotMobile '10, pages 43–48, New York, NY, USA, 2010. ACM.

[12] pandaboard.org. Open OMAP 4 processor-based mobile software development platform. http://pandaboard.org/, 5. August 2013.

[13] Aubrey-Derrick Schmidt, Rainer Bye, Hans-Gunther Schmidt, Jan Clausen, Osman Kiraz, Kamer Yüksel, Seyit Camtepe, and Albayrak Sahin. Static analysis of executables for collaborative malware detection on android. In *ICC 2009 Communication and Information Systems Security Symposium*, Dresden, Germany, Germany, 6 2009.

[14] Thomas C. Schmidt, Matthias Wählisch, and Michael Gröning. Context-adaptive Entropy Analysis as a Lightweight Detector of Zero-day Shellcode Intrusion for Mobiles. In *Poster at the ACM WiSec*, New York, June 2011. ACM. Poster.

[15] Thomas C. Schmidt, Matthias Wählisch, Benjamin Jochheim, and Michael Gröning. WiSec 2011 Poster: Context-adaptive Entropy Analysis as a Lightweight Detector of Zero-day Shellcode Intrusion for Mobiles. *ACM SIGMOBILE Mobile Computing and Communications Review (MC2R)*, 15(3):47–48, July 2011.

[16] Michael Spreitzenbarth, Sven Schmitt, and Felix Freiling. Forensic Acquisition of Location Data on Android Smartphones. In *Advances in Digital Forensics VIII*, Springer Science+Business Media, New York, USA, 2012. Bert Peterson and Sujeet Shenoi.

[17] Thomas C. Schmidt. Project-Homepage: SKIMS - A Cooperative Autonomous Immune System for Mobile Devices. http://www.realmv6.org/skims.html, 18. August 2013.

[18] Sebastian Trapp, Matthias Wählisch, and Jochen Schiller. Short Paper: Can Your Phone Trust Your Friend Selection? In *Proc. of the 1st ACM CCS Workshop on Security and Privacy in Mobile Devices (SPSM)*, pages 69–74, New York, 2011. ACM.

[19] Trusted Computing Group. TCG Mobile Trusted Module Specification. http://www.trustedcomputinggroup.org/files/static_page_files/3D843B67-1A4B-B294-D0B5B407C36F4B1D/Revision_7.02-_29April2010-tcg-mobile-trusted-module-1.0.pdf, 29. April 2010.

[20] Trusted Computing Group. TPM Main Part 1 Design Principles - Specification Version 1.2. http://www.trustedcomputinggroup.org/resources/tpm_main_specification, 1. March 2011.

[21] Trusted Computing Group. Developers - Mobile. http://www.trustedcomputinggroup.org/developers/mobile, 4. August 2013.

[22] Trusted Computing Group. TCG Trusted Platform Module Library Part 1: Architecture. http://www.trustedcomputinggroup.org/files/static_page_files/7F7F6AFE-1A4B-B294-D0EE43535A6176B2/TPM%20Rev%202.0%20Part%201%20-%20Architecture%2000.96%20130315.pdf, 15. March 2013.

[23] Trusted Computing Group. Trusted Computing Group - Home. http://www.trustedcomputinggroup.org/, 4. August 2013.

[24] Osman Ugus, Martin Landsmann, Dennis Gessner, and Dirk Westhoff. A Smartphone Security Architecture for App Verification and Process Authentication. In *21st International Conference on Computer Communications and Networks (ICCCN'12)*, pages 1–9, Munich, Germany, August 2012. IEEE.

[25] Osman Ugus, Dirk Westhoff, and Hariharan Rajasekaran. A leaky bucket called smartphone. In *4th Workshop on Security and Social Networks (SESOC'12)*, pages 374–380, Lugano, Switzerland, March 2012. IEEE.

[26] Matthias Wählisch, Sebastian Trapp, Christian Keil, Jochen Schönfelder, Thomas C. Schmidt, and Jochen Schiller. First Insights from a Mobile Honeypot. In *Proc. of ACM SIGCOMM, Poster Session*, pages 305–306, New York, August 2012. ACM.

[27] Matthias Wählisch, Sebastian Trapp, Jochen Schiller, Benjamin Jochheim, Theodor Nolte, Thomas C. Schmidt, Osman Ugus, Dirk Westhoff, Martin Kutscher, Matthias Küster, Christian Keil, and Jochen Schönfelder. Vitamin C for your Smartphone: The SKIMS Approach for Cooperative and Lightweight Security at Mobiles. In *Proc. of ACM SIGCOMM, Demo Session*, pages 271–272, New York, August 2012. ACM.

[28] Matthias Wählisch, André Vorbach, Christian Keil, Jochen Schönfelder, Thomas C. Schmidt, and Jochen H. Schiller. Design, implementation, and operation of a mobile honeypot. Technical Report arXiv:1205.4778, Open Archive: arXiv.org, 2013.

[29] Yajin Zhou and Xuxian Jiang. Dissecting Android Malware: Characterization and Evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 95–109, Washington, DC, USA, 2012. IEEE Computer Society.