

ObDroid: An Android permanent monitoring application using the observer pattern

Mihai Hornariu¹ Alexandru Butean²

¹Babes-Bolyai University; Faculty of Mathematics and Computer Science
1 M. Kogălniceanu Street, Cluj-Napoca, 400084, Romania
E-mail: hornariu.mihai@yahoo.com

²University “Lucian Blaga” of Sibiu; Faculty of Engineering
10, Victoriei Bd., Sibiu, 550024, Romania
E-mail: alexandru.butean@ulbsibiu.ro

Abstract. Nowadays, mobile devices are the most important gadgets from our lives. We use smartphones and tablets for many activities: communication, instant messaging, emails, multimedia content sharing, social networks, e-banking procedures, etc. The fact that we are already used with their advantages makes them indispensable in our daily activities. In this paper, we present a solution, an experimental application based on the observer pattern, a method to permanently monitor the main activities (calls, messages, location) using an Android device. The interaction between the analyzed device and the observer is possible using an intermediary server that makes the connection between a hidden service and the monitoring platform. Using a tool with monitoring capabilities can be extremely useful in many cases such as industrial use for employees, the family purpose for parental control, personal security as an anti-theft solution.

Keywords: Mobile devices; Android; observer pattern.

1. Introduction

Modern mobile devices have a major impact on people's lives. The most important fields in which mobile devices play a crucial role include industrial technology and family. The growth in the number of downloads of applications for smartphones and tablets has led to the development of mobile applications which store personal information about the user and his activity. The high degree of accessibility of such data can cause undesired effects like:

- Endangering privacy;
- Preventing the user from fulfilling his activity;

- Loss of important private data;

In order to prevent or reduce the impact of these hazards from taking place, sending an important amount of information to a person who will monitor the user is considered essential. The importance of this action is a major aspect that will be taken into consideration.

ObDroid is an experimental Android application that reacts whenever an event is triggered by the operating system on the device. This is the reason why the observer pattern represents the core of the design. In order to test whether this approach is reliable, the study presents the architecture and implementation steps for an early prototype. The result offers the end user the facility to observe the activities carried out by the supervised person, allowing both real-time and data history monitoring.

2. Related Research

2.1 Mobile Activity Monitoring System Using Android Spy

Since every company has unique policies, rules and Intellectual Property Rights, in some cases the privacy, security, and confidentiality must be maintained by the employees. Also, the usage of company's assets (mobile phone) for personal use or crossing out the organization's geographical area in working hours is not allowed. So, it is important to have a tool to track mobile phones to be able to implement such rules. The main advantages of the described system are: easy to use and track devices, multiple users can be tracked in the same time, provides security to find data leakage and security breaches in the organization works using notifications and instant alerts to the administrator.

2.2 Unsafe Exposure Analysis of Mobile In-App Advertisements

Almost two-thirds of the free mobile applications incorporate a library of advertising in their code. A study (Grace et al, 2012) observes, identifies and classifies these libraries and what risks they expose in terms of user privacy and security. These libraries collect private information (such as user location for further promotion), or some invasive information related to the call log or the message name, phone number or his account information.

ObDroid: An Android permanent monitoring application using the observer pattern 27

In order to prevent detection from such initiatives, ObDroid was developed without using any classical advertising libraries.

2.3 Cloud-based Real-time location tracking and messaging system: A child-care case study

Cloud Computing synchronization is one of the ways to achieve real-time monitoring over mobile devices to supervise various activities. Through mobile cloud services parents can watch their children and know if they are safe and sound. To achieve this, a cloud-based system CRLTMS (Huynh et al, 2015), reports data to a supervisor and offers the following features: finding a device location in real time, fencing notification, messages and indicators. Its purpose is to keep kids safe and report parents their activity.

3. Similar market applications

ObDroid, as an experimental proof of concept application, was developed using standard software engineering patterns. Keeping things simple and using low-level logic, a malware-like application acts undetectable on a target phone and proves how easy is to steal information from one of the most used OS around the world. While the research was ongoing, the market has already exploded and there are few good commercial applications that took the concept to the next level.

M spy

M spy offers a complete and user-friendly solution for mobile & computer monitoring activities. Hosts a platform that allows complex management scenarios of multiple devices. Each device must follow an installation and then it works as a set it and forget it service. Using standard advertising and spy libraries, the product offers a rich variety of features and maybe that is why it is easily detectable even by the free antivirus monitoring applications.

TrackmyFone

Whether is iOS or Android, this service gives an observer the ability to install a spyware application on a target and monitor its activity. Undetectable by most free antivirus solutions, this product offers a very

limited set of functionalities and remains one of the few market ready applications that can be used on any related use-case.

4. The Observer pattern

The Observer pattern is included in the behavioral design pattern family according to Kanasz (2013). As also stated by Taylor & Ray (2014), the main role of a behavioral pattern is to explain how objects interact. It describes how classes or objects communicate with each other in order to perform a task and how the steps of this task are divided among these objects (further divide it into smaller tasks allocated to corresponding objects) as stated by Pankaj (2016).

The use of the observer pattern is mainly described by a one-to-many dependency between objects so that when one object changes its state, the objects that depend on it will receive notifications and will be updated. Mainly, the observer pattern is useful whenever the developer is interested in the state of an object and wants to be notified if there is any change to that instance.

There are two main actor classes in the observer pattern. The object that watches over the state of another object is called the Observer, while the object that is being watched is called the Subject. The subject contains a list of observers to be notified if any change happens to its state. A subject should also provide methods that would be used by observers in order to register and unregister themselves.

An implementation of the observer design pattern is described in Figure 1. The SubjectBase is the class for all the subjects. It includes a protected list of observers that are subscribed to the respective subject. It also contains a list of methods that allow adding or removing observers and the Notify method which loops through the registered observers calling their own Notify method, as in Balkar (2014). The ConcreteSubject represents the concrete implementation of the SubjectBase class which maintains its own state and calls the Notify method as described above.

The ObserverBase class is the abstract class for all the observers, containing the method called whenever the state of a subject changes.

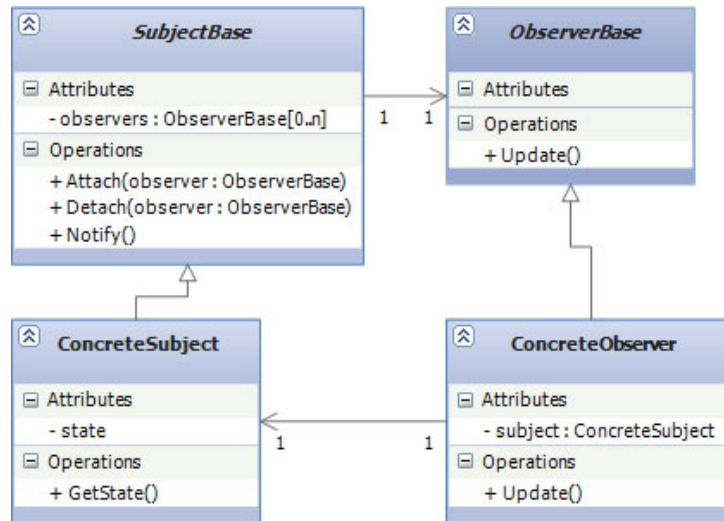


Figure1. UML diagram - observer pattern implementation

The **ConcreteObserver** is the concrete implementation of the **ObserverBase**, containing the subject to whom the observer is registered, and the implemented `Update` method.

5. Android's broadcast receiver

Broadcast receivers are important parts of any Android application. More accurately, a broadcast receiver is an observer pattern playing the role of a guardian or watching over the entire application. Inside the application, also known as a publisher, broadcasts can be generated to send events, not knowing if anyone will be noticed about the occurrence of these events, similar with the aspects related by Shedge (2013).

Receivers, also known as subscribers, fulfill the role of forcing the information to be subscribed by making use of filters. If there is a message matching a filter, that subscriber will be activated and notified of the message's presence as described by Balkar (2014).

In order to get notified, similar with the approach from (Huynh et al, 2015), whenever an action takes place, an application will have to subscribe to a **BroadcastReceiver**. The action is in the form of an intent broadcast. The receiver is woken up and will start executing whenever a matching intent is

launched. The “wakeup” or alarm event will happen in the form of a `onReceive()` callback method (seen in Figure 2), as Sarkar (2016) states.

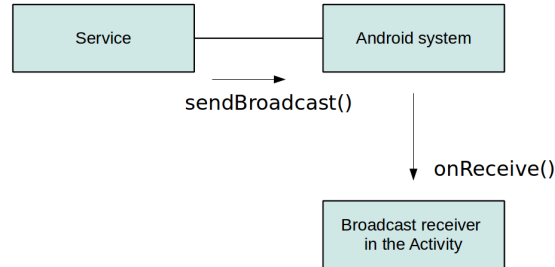


Figure2. Diagram of a Broadcast receiver being notified by a broadcast sent by a ServiceContent Observer

6. ContentObserver

The pattern of observing a dataset labels the various situations when a component makes the changes to the set and then invokes a notification interface so that the dependent components are aware that it has changed. (The dataset behavior pattern includes notifications every time a change is made to the dataset, alerting the corresponding dependent components.)

The Android content provider framework has an elegant design. URIs uniquely identify the datasets by asking for notifications when a certain URI has changed. The observer object is a child of the `ContentObserver` class as used by Wolfram (2015), so its `onChange()` method should be called by a content resolver if there is any existent change of the data behind the URI, the observer was registered for.

7. ObDroid architecture

In the case of a phone call, the data that will be collected on the Android Device and sent to the server includes the phone number, contact name, duration of the call. If we refer to a message, the text will then replace the duration and if the location is being taken into consideration, the address, the latitude, and longitude as well as the timestamp will also be sent as pieces of information to the server.

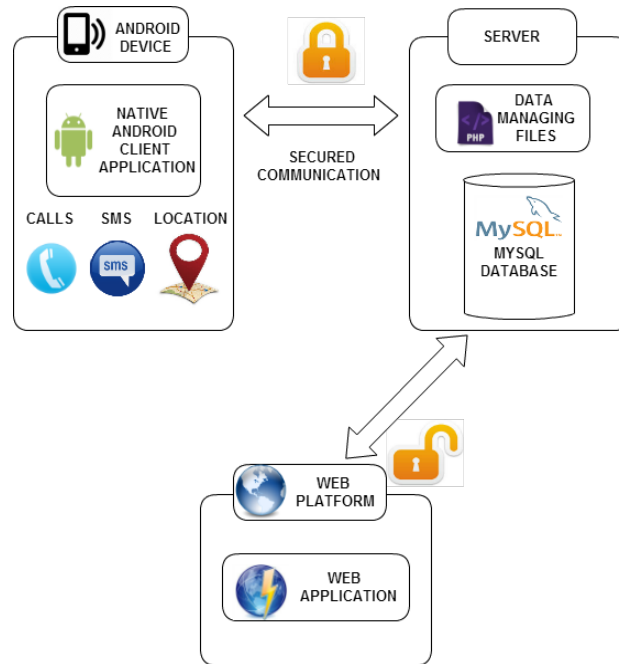


Figure 3. ObDroid Architecture

The information about phone calls, SMS and Location obtained inside the BroadcastReceivers and ContentObservers is sent to the server by a HTTP POST method. The data that will be sent through the POST method is encrypted using the Advanced Encryption Standard(AES) as in (Pahal et al, 2013), Cypher Block mode, with no padding as recommended by Dworkin(2001). A response from the server is being sent back to the Android Device to notify whether the POST was successfully delivered or not. The PHP scripts on the server are used to store this information in a database. In order to display the information in the Web application, it will be read from the database and decrypted using the same standard that was used during the encryption phase. The entire communication protocol can be observed in Figure 3.

Starting a service at boot

The first question that should be answered is if a broadcast receiver can be triggered before the application that it belongs to is being launched. If we take a closer look at the vulnerabilities of this pattern, as Tian (2016)

explains, the answer is affirmative, and the most clarifying example is given by triggering a receiver whenever the `BOOT_COMPLETED` event is registered as stated by Darwin (2016). This way, a service or an activity can be started by that receiver at the startup of the device. To successfully complete this task, permission for receiving the completed boot operation must be required and included in the manifest as described in Manifest File in Android, (2016).

Restarting a service whenever being closed

In order to avoid a service from being stopped from properly working when a user tries to destroy or close it, that service sends a broadcast including a message that will be observed by the matching `BroadcastReceiver`. When the `BroadcastReceiver` will be notified of the new message's presence, it will restart the service by invoking a restart method.

Call detection receiver

Whenever a phone call occurs, an action will be completed by an instance of the `BroadcastReceiver` class. This receiver will then be notified the moment when an action will change regarding the call state of the phone: from idle to ringing, from ringing to off-hook, vice versa scenarios.

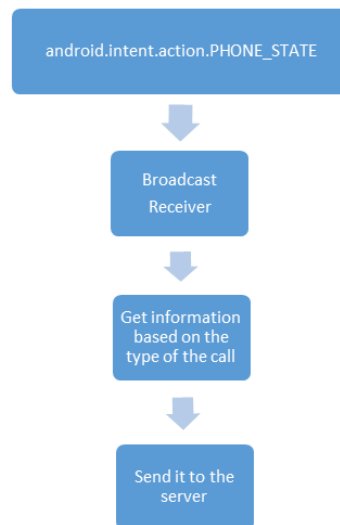


Figure 4. Call Process diagram – Call detection

In order to complete this type of action, a permission to read the phone state must also be included in the manifest file as described in Manifest File in Android, (2016). Different call types and information about the caller can now be used, based only on the transition between the three call states of the device. As seen in Figure 4, when a call is sent or received, the BroadcastReceiver gets notified about the state. Based on this notification, the call data can be accessed from the app and then sent to the server for storage. Using this approach, the device sends a log about the call details each time its state changes so it gets missed calls, received calls, dialed calls and rejected calls.

BroadcastReceiver for location

The location services need a considerable amount of battery power. This problem is also studied by Chis & Harrison (2016) and mentioned by HowtoGeek (2016). In order to reduce this impact on the battery, we concluded that the location of the device should be obtained every 30 minutes so that the battery consumption level stays decent. To get a precise location, these services need to be activated two minutes before the location is actually obtained. In order to fulfill this goal, an alarm is set to be woken every 30 minutes, and this alarm will send a broadcast to a class which has the functionality of polling the location of the device.

When the polling is completed, a broadcast containing extra information (the current location) is sent, to notify the BroadcastReceiver which will receive the final location data and use it further. This last receiver will decode the location data received and then will send it to the server. The permission which is required to access the device's location is called ACCESS_FINE_LOCATION and it must also be placed inside the manifest file.

Text messages detection

BroadcastReceiver cannot be used to detect whether a text message is sent or received because there is no broadcast throughout the system at that moment.

A Content Resolver instance is created, and a ContentObserver is registered to the resolver previously created, to listen for the changes that happen to the SMS content of the device. The ContentObserver will then integrate the functionality of querying the URI that contains the data parsed

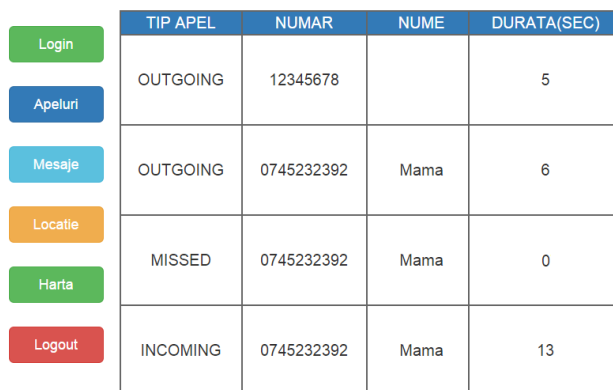
from the SMS content. With every change that happens to the respective content, the information about the latest message will be processed and sent to the server. The permissions needed to achieve this functionality include reading, sending and receiving SMS permissions.

8. Resulted work

8.1 Prototype

As a first working prototype of ObDroid, the application manages to extract data from the target's phone and send it to the server and then displayed for the supervisor.

An example, as shown in Figure 5 and 6, the information about the calls and location are extracted, placed in the database on the server and displayed in the table structure.



TIP APEL	NUMAR	NUME	DURATA(SEC)
OUTGOING	12345678		5
OUTGOING	0745232392	Mama	6
MISSED	0745232392	Mama	0
INCOMING	0745232392	Mama	13

Figure 5. Call information

Currently, the entire concept is based on a 1 to 1 link, where a viewer can only monitor one device because the extracted information is encrypted using a key generated based on the IMEI of the 2 phones: observer and target.

ADRESA	LATITUDINE	LONGITUDINE
Strada Gheorghe Lazăr 6-8 Sibiu	45.7955718	24.1542247

Figure 6. Localization information

8.2 Installation and Functioning

In order to successfully use the Android application, the beneficiary has to install the application on a target device and then follow a simple register procedure with his email account. A password will be sent to the user through the email service.

Whenever a call or message is received or sent, or the location is updated, these changes can be seen by the observer on the web application. The user has to log in with the email account and the password received at the registering step. He will be able to see a complete history of the activities performed on the monitored device.

The Call module includes information about the dialed number, name of the person (if exists as a contact), duration and timestamp for each call registered on that device. The SMS module includes extra information about the text of the SMS and the Location module provides data about the address of the device, the latitude, the longitude, and timestamp. All this information is sorted in reversed chronological order so that the user can see at first the most recent recorded activity.

8.3 Usage scenarios

The application offers the observer the possibility of monitoring the activity of another person.

The solution can be used by parents if they want to receive information about the activity of their children in a proper way. The location of a child can be detected at a certain moment in time. Also, the content of their text messages and the information about their phone calls can be monitored in real time.

Another area where this type of applications can be used is represented by the transport industry. A company that delivers packages or cargos would monitor the activity of their employees. The person that uses the application will be able to find out which are the messages the delivery agent sent to the customers and which are the calls made by the agent. These pieces of information, correlated with data about the employee's location, current date and time, will represent a relevant criterion for analyzing the activity of the employee.

8.4 Tests and issues

The installed application was tested on the following Android versions: 4.4, 5.0.2, 6.0.1. On versions 4 and 5, all the functionalities were working with no problems, but on version 6, the security updates prevented the deployed app from sending information about call logs while location and messages were not affected.

Also, in terms of messages, if the targeted phone has an external app that manages all messages (Facebook or WhatsApp) the messages will not be extracted at all because those applications are overlapping the standard SMS protocol using their proprietary protocols.

At this early stage, there were no tests concluding the amount of battery drainage caused by the installed application on the target's phone. Regarding the data traffic, the amount of information is directly influenced by the activity on the target's phone, measuring no more than a few Kb of data per day.

On the target phone, several antivirus applications were installed before and after the installation of the application. None of them reported a malicious activity coming from ObDroid's service. This achievement proves that the method described in this paper is valid and undetectable.

However, in a more extreme testing environment, if a firewall is installed using root rights and has an active professional feature to disable all internet access without confirmation, then for every information sent through the existing channels ObDroid will request a confirmation. This extreme case applies to all other applications on the device and can block the interface if there are many applications requesting access to once: messenger, weather, location, browser, social media, etc. This case is highly improbable but while the issue was discovered a solution was also identified at a conceptual level: while root user is performing on the device, do not send any data.

8.5 Future research and development

Considering the very early stage of this concept, soon a variety of other features will be added to the initial concept, some of the most important upgrades are:

- Offline usage: the application does not cover the situation where the user is offline. To allow monitoring while being offline, a mechanism must store the data in a local encrypted file until the device has a working internet

connection and send it afterward to the server.

- Fencing: a safe area space, a fencing feature could notify the observer whenever the monitored subject leaves a pre-established area (in the form of a circle on the map).
- Battery drainage measurement and corresponding settings to the observer to reduce the power consumption when necessary

9. Conclusion

In this paper we presented a mobile activity monitoring tool that is taking advantage of several Android vulnerabilities, similar to the ones described by (Hamandi et al, 2016), enabling a person to monitor another person with an Android device by using a previously installed hidden application that communicates with a server using a protocol that complies with the observer pattern available in the operating system's SDK. The information (calls, messages, location) can be accessed on an external platform that communicates with the server. During the communication process, all the personal data is encrypted and each monitored device has its own personal generated encryption key so that only a specific observer can get access to a set of information.

References

- Balkar, K., (2014), *Detect incoming call and call hang-up event in Android*, Retrieved on 5.11.2016 from <http://karanbalkar.com/2014/02/detect-incoming-call-and-call-hangup-event-in-android/>
- Chis, T., Harrison, P., (2016), *Performance-energy trade-offs in smartphones*, 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, At Malta, Malta, DOI: 10.1145/2988287.2989140
- Darwin, I., (2016), *Android Cookbook, Problems and Solutions for Android Developers*, O'Reilly Media
- Dworkin, M., (2001), *Recommendation for Block Cipher Modes of Operation*, NIST Special Publication 800-38A
- Grace M., Zhou W., Jiang X., Sadeghi A., (2012), *Unsafe Exposure Analysis of Mobile In-App Advertisements*, WISEC '12 Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks, 101-112
- Hamandi, K., Alaa, S., Elhaji, I., Kayssi, A., (2015), *Messaging Attacks on Android: Vulnerabilities and Intrusion Detection*, Mobile Information Systems, DOI: 10.1155/2015/746930

- HowtoGeek, *Why is Google Services Draining So Much Battery on Android?* (2016) Retrieved on 28.10.26 from <http://www.howtogeek.com/193982/why-is-google-services-draining-so-much-battery-on-android/>
- Huynh, C-T., Nguyen, H-Q, Pham, X-Q, Nguyen, T-D, Huh, E-N, (2015), *Cloud-based Real-time location tracking and messaging system: A child-care case study*, published in *ACM International Conference on Ubiquitous Information Management and Communication*, ACM IMCOM, ISBN: 978-1-4503-3377-1
- Jagtap N., Patil K., Shakil S., Ingle N., (2015), *Mobile Activity Monitoring System Using Android Spy*, *International Journal of Advanced Research in Computer and Communication Engineering*, Vol. 4, Issue 2, ISSN: 2278-1021 ISSN: 2319-5940
- Kanasz, R., (2013), *Behavioral Design Patterns*, Retrieved on 20.11.206 from <http://www.codeproject.com/Articles/455228/Design-%20%20%20%20%20Patterns-of-Behavioral-Design-Patterns>
- Manifest File in Android*, (2016) Retrieved in 10.11.2016 from <https://developer.android.com/guide/topics/manifest/manifest-intro.html>
- Pankaj, (2016), *Observer Design Pattern in Java*, Retrieved on 2.11.2016 from <http://www.journaldev.com/1739/observer-design-pattern-in-java>
- Pahal, R., Kumar, V., *Efficient Implementation of AES*, (2013), *International Journal of Advanced Research in Computer Science and Software Engineering*, Volume 3, Issue 7, July 2013 ISSN: 2277 128X
- Sarkar, V., (2016) *Java Design Patterns, A tour with 24 Gang of Four Design Patterns in Java*, Apress Publishing House
- Shedge, K.N., Pathak, S., Rokade, S.M., (2013), *Android-Broadcast Receiver*, *International Journal of Emerging Technology and Advanced Engineering*, ISSN 2250-2459
- Tian, D., (2016), *Detecting Vulnerabilities of Broadcast Receivers in Android Applications*, Oshawa, Ontario, Canada
- Taylor, G., Wray. R., (2014) *Behavior Design Patterns: Engineering Human Behavior Models*, Behavior Representation in Modeling and Simulation (BRIMS)
- Wolfram, R., (2015), *Use Android's ContentObserver in Your Code to Listen to Data Changes*, *GrokkingAndroid*, Retrieved on 20.11.2016 from <http://www.grokkingandroid.com/use-contentobserver-to-listen-to-changes/>
- of Broadcast Receivers in Android Applications*, (2016), Oshawa, Ontario, Canada