

Design And Implementation Of An Android Based Botnet

Andreas Borgwart

Bachelor Thesis

Design And Implementation Of An Android Based Botnet

vorgelegt von

Andreas Borgwart

6.06.2012

Technische Universität Kaiserslautern
Fachbereich Informatik
AG Verteilte Systeme

Betreuer: Prof. Dr.-Ing. Jens B. Schmitt

Prüfer: Prof. Dr.-Ing. Jens B. Schmitt Dipl.-Technoinform. Matthias Wilhelm

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben. Alle wörtlich oder sinngemäß übernommenen Zitate sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Kaiserslautern, den 6.06.2012

Andreas Borgwart

*Ich danke Tim Stöber für seine kompetente
Unterstützung bei der Implementierung.*

Contents

List of Figures	iii
1 Motivation	1
2 Introduction	3
2.1 Terminology	3
2.1.1 Android	3
2.1.2 Exploit	4
2.1.3 Rootkit	4
2.1.4 Botnet	4
2.2 Goal Of This Thesis	5
3 Related Work	7
4 Botnet Design	9
4.1 Command And Control	9
4.2 Botnet Structure	9
4.2.1 Hierarchical	10
4.2.2 Peer-To-Peer (P2P)	10
4.3 Characteristics Of A Mobile Botnet	12
4.3.1 Advantages	12
4.3.2 Disadvantages	13
5 The Bot	15
5.1 Bot Implementation	15
5.2 Android Security	16
5.2.1 Component encapsulation	16
5.2.2 Application permissions	16
5.2.3 Application signing	17
5.2.4 Dalvik virtual machine	18
5.3 Structural Overview	19
5.4 Capabilities	21
5.4.1 Register and Home call	21
5.4.2 Intercepting SMS	21
5.4.3 Audio Recording	22
5.4.4 Environment Manager	22

6	The Botmaster	25
7	Evaluation And Prospect	27
7.1	Evaluation	27
7.2	Prospect	28
8	Conclusion	29

List of Figures

1.1	Mobile Threats by Platform, iOS (Apple), J2ME (Sun Microsystems), PocketPC (Palm), Symbian(Nokia), Android (Google), 2004-2011 . . .	2
2.1	Android System Architecture	3
4.1	Botnet Structure	10
4.2	Smartphone Platform Share	14
5.1	Bot Permissions	17
5.2	Android Version Distribution	18
5.3	Simplified UML Model	20
5.4	SmsReceiver As Declared In The Manifest	21
6.1	The Web Interface	26
7.1	Battery Use With DiscoBot	27
7.2	Battery Use Without DiscoBot	27

List of Figures

1 Motivation

Smartphones are continuously becoming more and more powerful. Their computational power increases while the battery still lasts as long as before or even longer. With today's mobile operating systems, such as Android (see section 2.1), Smartphones are able to perform almost every task a desktop computer could. However, they also have some attributes which makes them excellent targets for malicious purposes.

Firstly, they have built-in sensors which can be exploited to monitor the user's physical environment. Secondly, Smartphones normally are near their user and therefore able to record conversations, film and track location. Thirdly, it is likely that a Smartphone contains sensitive personal information like appointments and a wide range of contact information.

These attributes are the reason why Botnets (see section 2.1.4), as known from the personal computer (PC), are becoming more and more attractive to the mobile world. Already there have been some cases, for example a botnet called "Counterclank" which infected millions, stole private information and displayed ads [10]. This thesis will show that Mobile Botnets can be considerably more evil.

According to Google's Senior Vice President of Mobile, Android device activations were at 850,000 a day on 27.02.2011 [9] and this number is steadily increasing. Not only do device activations increase, but mobile threats for Android have rapidly increased since 2010, as a study from F-Secure shows.

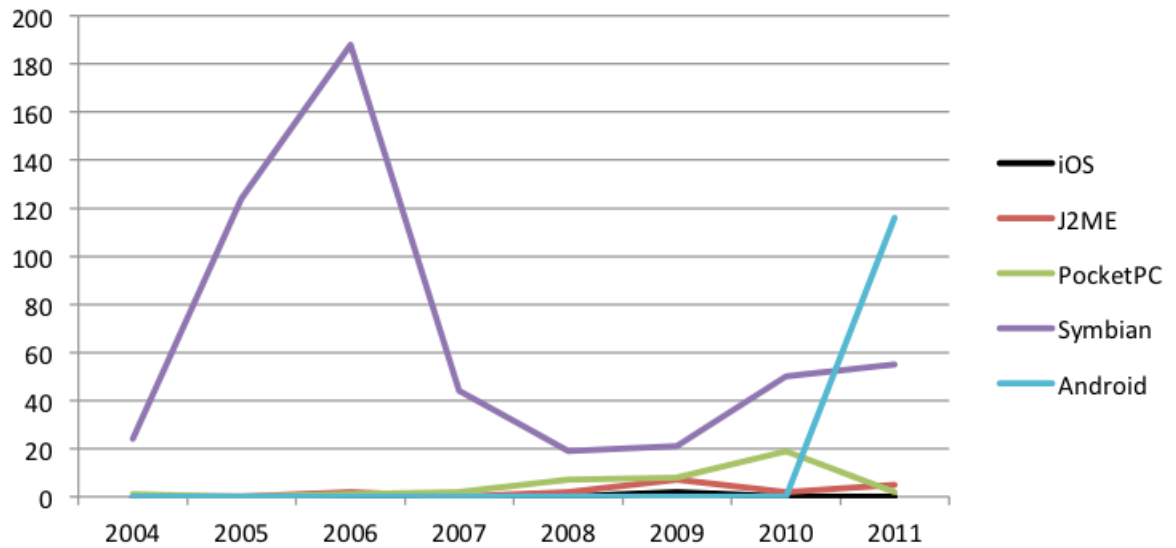


Figure 1.1: Mobile Threats by Platform, iOS (Apple), J2ME (Sun Microsystems), PocketPC (Palm), Symbian(Nokia), Android (Google), 2004-2011

[12]

Due to the described popularity of Android and the accompanying security concerns this thesis will give the reader an overview of Android Botnets. This includes design solutions, command and control structures, characteristics, capabilities and other related work. Furthermore, a proof of concept Botnet is implemented and explained.

2 Introduction

2.1 Terminology

2.1.1 Android

“Android is a software stack for mobile devices that includes an operating system, middleware and key applications.” [8]

Android can be used for mobile phones and tablets and has been available since October 2008. It is open source and developed by Google. The overall System Architecture is shown underneath.

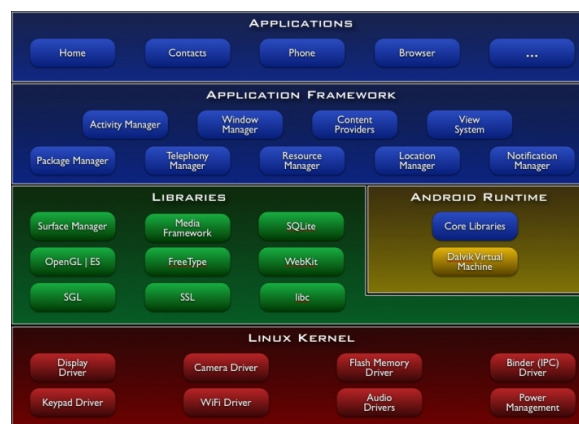


Figure 2.1: Android System Architecture
[8]

This graphic shows that the system can be logically divided into five layers: Applications, Application Framework, Libraries, Android Runtime and the Linux Kernel. All applications can be written in Java, which makes Android popular among developers, because Java is an easy to use and wide spread programming language.

Typically, as shown in the related work section, the phone is rooted first using exploits, see section 2.1.2, and only then the bot is installed in a lower layer. The bot created for this thesis is part of the normal application layer and is disguised as useful application. How it can work there stealthily, and simultaneously be used for malicious purposes even though it seems to be an ordinary application, is described in this thesis.

2.1.2 Exploit

An exploit is an attack which takes advantage of a known vulnerability in the target system. This often is the case if the system has not been updated and therefore is still vulnerable [18].

2.1.3 Rootkit

“A rootkit is a collection of tools (programs) that enable administrator-level access to a computer or computer network. Typically, a cracker installs a rootkit on a computer after first obtaining user-level access, either by exploiting a known vulnerability or cracking a password” [19].

2.1.4 Botnet

“A botnet is a group of computers connected in a coordinated fashion for malicious purposes. Each computer in a botnet is called a bot. These bots form a network of compromised computers, which is controlled by a third party and used to transmit malware or spam, or to launch attacks. A botnet may also be known as a zombie army” [23].

In this case the bots/zombies are Android phones. To compromise a phone its user has to install the Bot. More about the structure and control system of botnets is explained in section 4.

2.2 Goal Of This Thesis

The goal of this thesis is to implement an Android application with bot functionality and to propose different solutions to form a working botnet. No exploits are used and the phone does not have to be rooted for the bot to work properly. All there is to do in order to become a zombie of the botnet is to install the application which appears useful to the user but is in fact a hidden audio recorder and location tracker. The proposed botnet is designed to be a spy network in which the user's whereabouts are tracked and stored on a server and it is possible at any time to start the phone's secret audio recorder to gain private and sensitive information.

The Bot developed in this Thesis is called DiscoBot. Its source code can be found on the CD in the back of the thesis.

Different suitable structures of botnets are presented and analyzed with regard to their structure and their command and control mechanisms. Furthermore, the advantages and disadvantages of mobile botnets, in comparison to PC botnets, are analyzed.

Additionally, it is demonstrated that the implemented DiscoBot, once installed, can work stealthily with regard to command execution, audio recording, location tracking and battery drain.

Ultimately, it is a proof of concept that it would be feasible to build a spy network with an enormous database filled with precise information about each infected user. Thus it would be possible to have substantial user profiles with a list of their whereabouts collected every hour (or every X minutes), recordings of their phone calls and maybe even some secret audio information gathered for example during meetings. Furthermore, with the aid of these profiles it would be feasible to predict the whereabouts of the users.

3 Related Work

In this section, several different approaches to the design and control of a botnet and other related studies are presented.

In [27] Zeng et al. compared different P2P structures regarding their performance if used in a mobile botnet and came to the conclusion that a modified Kademlia is best suited. Kademlia and the modifications the authors made are further described in section 4.2.2. They, too, controlled their bots with SMS, but in contrast to this work, the message is visible to the user appearing as a spam SMS. That means they hide the commands in fake error codes or passwords and extract the real command. The authors claim that spam SMS are “familiar to today’s phone users” [27] and even if the user deletes the SMS it is no problem, because the command has already been executed directly after reception.

That mobile botnets can be used for traditional botnet denial of service attacks is demonstrated in [24]. The authors managed to successfully attack the cellular infrastructure, the Home Location Register (HLR) in particular, which is the central repository of user information required to establish every phone call. With constantly using the *insert_call_forwarding* command of 11.750 mobile phones to occupy the HLR the authors managed to reduce the legitimate service by 93% in an area-code sized region.

Other botnets are described in [15] and [25]. In both cases the phones need to be rooted before the bot can be installed. In the first paper, the authors hijacked already jailbroken iPhones in which the root password was not changed after the jailbreak¹. In the second, Georgia Weidman assumes that the phone is already rooted and focuses only on the structure and command and control mechanisms. The Bot she proposes lodges itself between the Application Layer and the GSM Modem to intercept incoming SMS. These are scanned for an identifier and then either passed on to the application layer, or kept and executed. This was the inspiration for the SMSReceiver in section 5.4.2, but since her bot is in a lower layer it had to be written in C. The advantage in doing so is that it is more portable, see 4.3.2, but the disadvantage is that it has to be a rooted phone to begin with.

In [16] the authors demonstrated that it is possible to write a rootkit for Android and wrote a sleeper application, meaning they chose a popular topic, in this case a fake *Twilight* application, to attract as many people as possible, and waited for a new vulnerability to become public. Then they just had to push down the payload containing

¹After a jailbreak the iPhone gets a default root password, http://en.wikipedia.org/wiki/iOS_jailbreaking

the malicious code to exploit all phones with their sleeper application. As soon as the phone was rooted, one of the bots above could be installed. Another Android toolkit was developed in [17].

A sound Trojan called “Soundcomber” was developed by Roman Schlegel et al. in [20]. It has the ability to extract credit card and PIN numbers. Soundcomber records every phone call and, by using speech recognition, locally analyzes the beginning of the conversation in order to determine if the user is interacting with a service hotline. Soundcomber was able to detect 55% of all hotline calls in their experiment with five different hotlines. If the credit card or PIN number is extracted, a paired application is used to stealthily send it to the master. Thereby only the extracted number is sent. Soundcomber does not concern itself with normal conversations, because they try to keep the communication overhead and the computation time to a minimum, hence the recording is immediately aborted if no hotline can be detected. As soon as Soundcomber decides that the current conversation is a normal call, the recording is discarded. They also provided a solution to prevent hidden audio recording which is described in section 5.1.

Naturally, the security researchers are not sleeping and for instance in [2] an improvement for the Android permission model was successfully developed, introducing finer grained permissions. Hence, an application requesting Internet access would only be allowed to connect to a specific set of hosts. Until now the Internet permission grants access to the entire Internet. Another security enhancement is the Kirin Framework developed in [3]. Kirin extracts the security policy from an application’s manifest to determine whether the requested set of permissions is harmful or not, using a logical engine (Prolog) to do so. Both Kirin and SCanDroid [4] will prevent a potentially harmful application from being installed. SCanDroid not only takes the manifest file into account, but also the code in order to analyze the new application’s flow of information together with the information flow of all installed applications.

4 Botnet Design

This chapter discusses different design solutions, i.e. the way commands and control messages are distributed, and possible network structures.

4.1 Command And Control

Computer based botnets usually use IP-based protocols to transmit command and control (C&C) messages and thus rely on the Internet as their distribution channel. Using an Internet connection with today's phones is feasible, but not advisable for the the following reasons. Mobile phones do not have a public IP address and therefore would have to use a centralized network approach, creating a single point of failure [25]. Other disadvantages of using IP-based C&C messages are the availability of steady Internet connections of a mobile phone, and that these connections are more power consuming than SMS messages [27].

As a result bots in this thesis are controlled using SMS messages. The advantages [15], [27] of using SMS messages are:

1. SMS is always on. As long as the phone is turned on it can receive SMS messages.
2. The phone number represents a public IP address.
3. C&C messages can be distributed to offline bots, because SMS messages are stored at the network provider and will be delivered as soon as the bot comes online again.
4. Especially in Android, Internet usage is visible in the status bar, but SMS messages can be delivered and processed stealthily, see 5.4.2.
5. As mentioned before SMS consumes no additional power.
6. They are hard to observe by security researchers.

4.2 Botnet Structure

There are two different approaches to design the botnet. The first one is to organize the bots hierarchically in a tree structure with centralized control. The second one is a decentralized peer-to-peer fashion.

4.2.1 Hierarchical

In a hierarchical tree structure the zombies are organized like in the following figure.

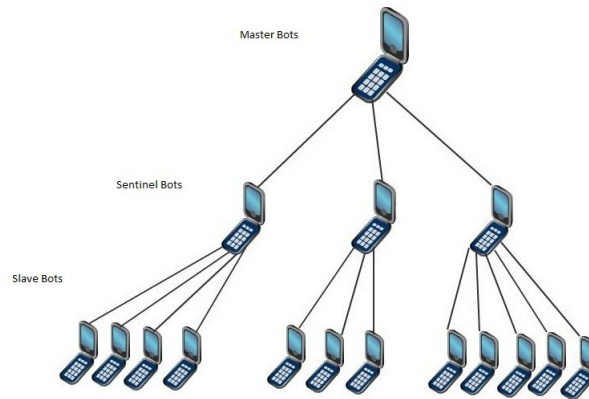


Figure 4.1: Botnet Structure
[25]

To distribute commands, the Botmaster sends them to all sentinel bots, who distribute them to their slave bots. By using more tiers of sentinel bots the amount of messages a single sentinel has to send can be reduced. The sentinel bots should be the most trustworthy phones. Therefore some information ought to be stored for each Bot, for example time since infection, availability, provider and battery life data. With this information bots can be promoted to sentinel bots or downgraded to slaves. It would be wise to group slaves by provider because SMS within the same provider is typically free, and therefore avoids detection from phone bills. Each bot only knows about bots, who are in a lower tier than itself, so only the Botmaster is aware of the full network. To secure the communication, an asymmetric cryptographic algorithm is used and each bot has the public key of the Botmaster. Elliptic curve cryptography (ECC) as described in [11] is suited best for this purpose due to the space limitations in SMS and computational restraints of the Smartphone.

4.2.2 Peer-To-Peer (P2P)

A pure peer-to-peer network is a network in which every peer can act as client or as server for other peers. P2P networks are very robust against some attacks and are harder to analyze [27]. For example, if some peers are the victim of a denial of service attack in a Kademlia network, the others would automatically route around them [13]. Therefore the C&C channel would stay intact.

In a hierarchical structure the failure of a sentinel bot results in the disruption of the C&C channel to his slave bots. Additionally, attackers might gain information about these slave bots or find a trace to the Botmaster if they compromised a sentinel bot.

A disadvantage, especially in a mobile botnet, is the larger message overhead to control a P2P network. This is why the Kademlia protocol in [27] was modified.

There exist many different P2P structures which could be used to create and control a botnet. They can be divided into three categories:

1. **Centralized**

A central server is needed to organize and maintain the system and therefore this structure has a single point of failure.

2. **Decentralized but structured**

These P2P networks use a network-wide consistent protocol to ensure that an efficient route is found to a desired content. Contents are stored at specific nodes and typically a Distributed-Hash-Table (DHT) is used, in which (key,value) pairs are stored in order to assign ownership of files. Kademlia represents an implementation of this type and is described below.

3. **Decentralized and unstructured** In this kind of network peers have to flood the network with queries for a specific content to find the owner, because there are neither central directories nor is there a protocol to manage content placement. Due to the flooding unstructured peer-to-peer networks have severe scalability issues.

In this thesis a centralized structure is used for testing, despite the drawback mentioned before, of a single point of failure. However, if someone would like to use DiscoBot in a real life scenario, he or she should use Kademlia as a network structure, because, in connection with SMS to deliver command and control messages, it is a good choice in terms of message overhead, delay, and load-balancing, as the study in [27] shows.

Kademlia: Kademlia is a peer-to-peer storage and lookup system [13]. It defines the network structure, and the peers form an overlay network using a DHT. Each node has a unique 160 bit *node ID*. The *distance* between two nodes is defined as the bitwise *exclusive or* (XOR) of their IDs. If a node publishes a file it first uses a hash function to compute a 160 bit hash value of the file and then searches for several node IDs in the network which have a short distance to the hash value. To these nodes the contact information of the publishing node are transmitted. Nodes searching for a file do exactly the same. They use the hash value to find nodes which have the contact information of the node where the file is stored. The hash function has to be the same across the whole network.

Kademlia routing tables are lists for each bit of the node ID. These list are called *k-buckets* and can have a maximum of *k* entries (for example $k=20$). A *k-bucket* contains the contact information of other nodes depending on the distance to them, for example if the distance to a node is *n*, then the contact information would be stored in the n^{th} *k-bucket*. Every node encountered in the network is considered for inclusion in a *k-bucket* using a specific algorithm.

There are four types of remote procedure call messages in Kademlia:

1. **PING** checks whether a node is available.
2. **STORE** is a request to store a key on a node.
3. **FIND_NODE(ID)** returns nodes closest to the ID of the argument.
4. **FIND_VALUE(KEY)** returns nodes closest to the ID of the argument.

In a mobile botnet some slight modifications improve the message overhead of Kademlia [27]. By using SMS messages to transmit command and control messages the PING command is unnecessary, because SMS messages will reach offline targets as soon as they come online again. Eliminating the PING command reduces the overhead, reduces costs and thereby increases stealthiness. The second modification affects the creation of node IDs. Instead of randomly creating them, the ID is a hash value of the phone number. This provides some security against node replication attacks where attackers try to infiltrate the network by adding new nodes in order to gain knowledge about the network, or to disrupt command and control messages.

4.3 Characteristics Of A Mobile Botnet

In a mobile botnet the characteristics of mobile phones have to be taken into consideration. This leads to the following advantages and disadvantages.

4.3.1 Advantages

The greatest advantage of a mobile botnet is that almost every device has accessible built-in sensors. This allows observation of the physical surrounding of the phone's user and to record for example audio, video [26] and location information.

Another great advantage is that, in contrast to a desktop computer, a mobile phone is most likely to be where the user is, even in very private places. This gives the opportunity to gather very sensitive and private audio recordings, for example, during meetings or confidential conversations. DiscoBot is able to record audio at any time, assuming the phone is connected to the network/Internet and the Botmaster is able to listen in almost real-time if using streaming mode. Due to location tracking it is also feasible to start recording automatically if the user is near a location of interest, for example a bank, or a governmental building.

A further advantage is that "the GSM modem [through which SMS messages are sent] can be viewed as a public IP address without filtering or firewall capabilities" [25] and SMS is an *always on* technology. As a result every valid SMS will be delivered if the recipient's phone is connected to the network or will be queued until it is. Additionally SMS messages are processed without user involvement which all in all makes "SMS the perfect attack vector against mobile and smart phones" [14]. This is why SMS is used to secretly control DiscoBot.

4.3.2 Disadvantages

Using a botnet consisting of mobile phones has several disadvantages compared to a traditional botnet of desktop computers. [15]

- **Stealthiness**

In order to stay hidden it is crucial that a bot does not drain the battery excessively and that it avoids a high CPU workload which would cause the user interface to lag. Otherwise the user probably will investigate the cause and might find and delete the bot or reset his phone. It is also important that the recorded files, or generally any gathered information, are unreachable or unusable for the user. Therefore DiscoBot stores the recordings in its own application space which is inaccessible for the user. That is a nice property of the Android system, but it is only stealthy on non rooted devices. On a rooted device it is possible to browse through the application space.

Another possibility for hiding files would be to split them in small parts and change their file extensions to something the system can not work with, as they did in [26]. This would only be necessary if the internal storage space were full or if the device were rooted. With this method, DiscoBot can store the recordings in its internal space or on the SD-card.

- **Communication Cost**

Unless the user is currently having a flatrate for SMS or Internet usage, it will cost him money. That means it will alert the user if the bot is significantly raising the monthly cost. As a result DiscoBot only sends the audio files automatically if the transmission is free, that is, if connected to a WLAN access point.

- **Performance**

A mobile phone has less computational power than a desktop computer which must be taken into consideration for the tasks a bot is performing. DiscoBot is not doing any sophisticated computations and audio recording, as well as Internet usage, runs smoothly in the background without the user's noticing.

- **Connectivity**

The connection speed and availability vary greatly depending on the user's location and the connection type. When the phone is connected to a WLAN the connection does not differ from a desktop's, but it is more likely to be slower with a more or less unpredictable availability. For DiscoBot, this is of no importance, because, unless in streaming mode, it is not time critical when, or how fast, the recordings are transmitted.

- **Platform Diversity**

In the mobile world, there are many different devices and platforms which makes it harder for a mobile botnet to grow as big as its desktop counterpart. The major platforms are Android, iOS, Windows Phone 7 and Blackberry.

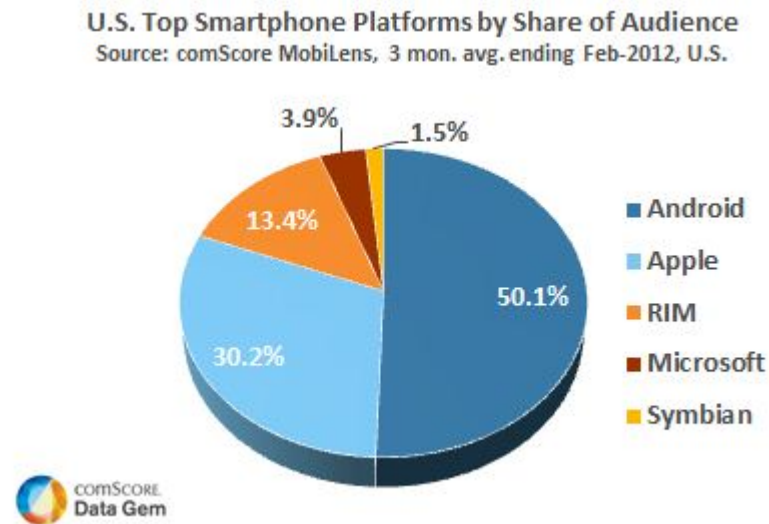


Figure 4.2: Smartphone Platform Share
[25]

If the bot requires a root access to the phone, it depends on the platform as well as the particular device to find an exploit which limits the scope of the botnet. It generally applies that: the bigger the botnet, the better. That is why attackers focus on the biggest platforms. A market share of 50% makes Android an worthwhile target.

5 The Bot

5.1 Bot Implementation

As mentioned before all applications for Android are written in Java, so for the implementation, Eclipse and the Android SDK (API 10 / Android 2.3.3- 2.3.7) was used in combination with two *Google Nexus S* Smartphones.

This work is only a proof of concept and therefore the application has, except a little audio recorder which can record and play audio, no functionality accessible to the user. To get users to download and install the bot it would be necessary to disguise it, for example, as a professional organizer.

This organizer would need certain functions to justify the requested permissions. In the following table the requested permissions are listed with a possible explanation why the permission is needed. Another advantage of hiding behind an organizer would be that the user presumably will use it to organize dates and times of appointments during which the AudioRecorder could be automatically started in the background.

Permission	Explanation
Location	save location of a meeting / geotag for voice memo
Send/receive SMS	automatically remind people of meeting
Internet	synchronization
Phone state	pause recording of voice memo at incoming calls
Record audio	record voice memo
Start at boot	needs to be always on to display reminders

The organizer might also request access to the address book to send reminder SMS messages, and then the address book could be sent secretly to the Botmaster.

Also, there have been some attacks on the permission model itself. In [1] they managed to establish an Internet connection without possessing the INTERNET permission and tricked the user to install an application.

5.2 Android Security

Besides the security mechanisms of the underlying Linux system there are some Android-specific ones: component encapsulation, application permissions, application signing and the Dalvik virtual machine [21],[5].

5.2.1 Component encapsulation

Every component of an application has an *exported* property which defines whether another application is allowed to use this component. If *exported* is set to *false* then only the owning application or an application with the same UserID (see Application signing in section 5.2.3) can access it. The Bot only implements components with *exported* set to *false* to prevent others from using the malicious functionality [22], [21].

5.2.2 Application permissions

“By default, an Android application can only access a limited range of system resources. The system manages Android application access to resources that, if used incorrectly or maliciously, could adversely impact the user experience, the network, or data on the device” [7]. To use a protected API, the application has to request the permission in its manifest. Before installation, a dialog with all requested permissions is presented to the user, who has to accept them in order to continue. It is not possible to accept only a subset of permissions. This leads to an *all or none* concept and it is likely that the user will continue even though he might have given only some permissions if he could. In the Android market 60% of all applications request access to the Internet [2]. This allows communication with any host and many applications only use Internet to download current advertisements. Permissions last as long as the application is installed. Once given, the user can not control when or how the application uses its granted permissions. For example, the RECORD_AUDIO permission: if granted, it is possible to record audio at any time without knowledge of the user unless his phone is in developer mode and he had carefully read the USB debugging log which is very unlikely.

A solution to prevent hidden audio recording is to implement a controller which determines if the call is sensitive or not. This can be used to blank out audio on demand for other applications, as done in [20]. Another solution is to use finer-grained permissions, for example an extra permission explicitly for in-call audio recording to raise the user’s awareness of the applications capabilities. [2]

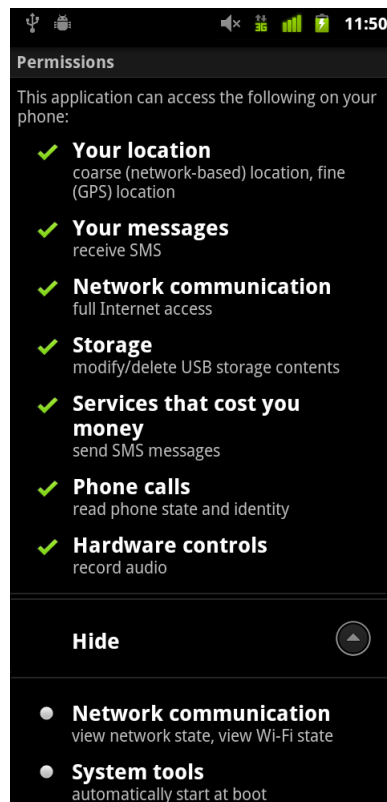


Figure 5.1: Bot Permissions

These are the permissions the Bot is using.

5.2.3 Application signing

Each application has to be signed, which can be done by the developer or a third party, in order to be accepted in the market place. If a developer signs two or more applications and explicitly requests a shared UserID in the manifest it is possible for these applications to access each other's components. During installation permissions are assigned to the UserID.

So a shared UserID possesses a combination of all given permissions without the user's explicit approval.

A possible attack scenario is that two applications sharing a UserID use their combined permissions for malicious purposes. One could request the Internet permission and the other one the permission to access the contact list. After that it would be possible for both applications to send contacts over the Internet due to the shared UserID. This scenario is further described in [22].

5.2.4 Dalvik virtual machine

Each application runs in its own Dalvik virtual machine, but this security mechanism is of no concern, because the presented bot is a regular application with granted permissions. However, it is important for exploits, because the installed application has to break out of it in order to gain root access.

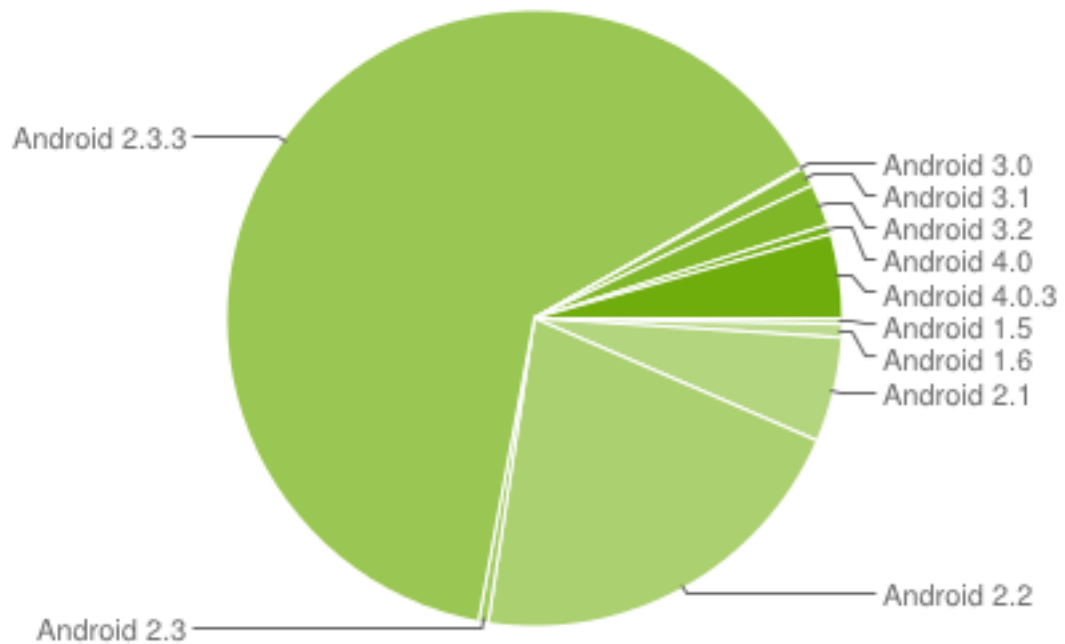


Figure 5.2: Android Version Distribution
[6]

The most wide spread Version still is 2.3.3 with 63,9% which is the one DiscoBot is developed with. It is advisable to develop an application on older versions due to compatibility reasons and because there might already be some exploits known which are only fixed in newer versions.

5.3 Structural Overview

In this section, details about the implementation are described with the help of a reduced UML diagram. Android components use *intents* to communicate with each other, so DiscoBot uses the already existing communication methods of the underlying system. For example, if the system receives an SMS it uses an *Broadcast Intent* which will be received by all receivers who are listening to *SMS_RECEIVED* actions. The *SmsReceiver*, see 5.4.2, extends an Android *BroadcastReceiver* to receive incoming SMS. During the installation DiscoBot registers the following receivers:

- **StartUpReceiver**

The *StartUpReceiver* listens to the *BOOT_COMPLETED* action so that DiscoBot will be started after the phone is booted. Therefore DiscoBot is always running in the background.

- **SmsReceiver**

The *SmsReceiver* listens to the *SMS_RECEIVED* action, which is described in detail in 5.4.2.

- **HeartBeatReceiver**

The *HeartBeatReceiver* is not listening to a specific action, but rather to an internal *intent* periodically created by the main activity. A timer creates an *Identifier* object at hourly intervals. The *Identifier* then gathers information about the phone and either registers it or updates the database of the Botmaster, see 5.4.1.

- **CallInterceptor**

The *CallInterceptor* listens to *PHONE_STATE* actions which are either *ringing*, *off hook* or *idle*. In case of *off hook* a new *AudioRecorder* will start both for incoming and outgoing calls, and if the phone state changes to *idle* the *AudioRecorder* is stopped.

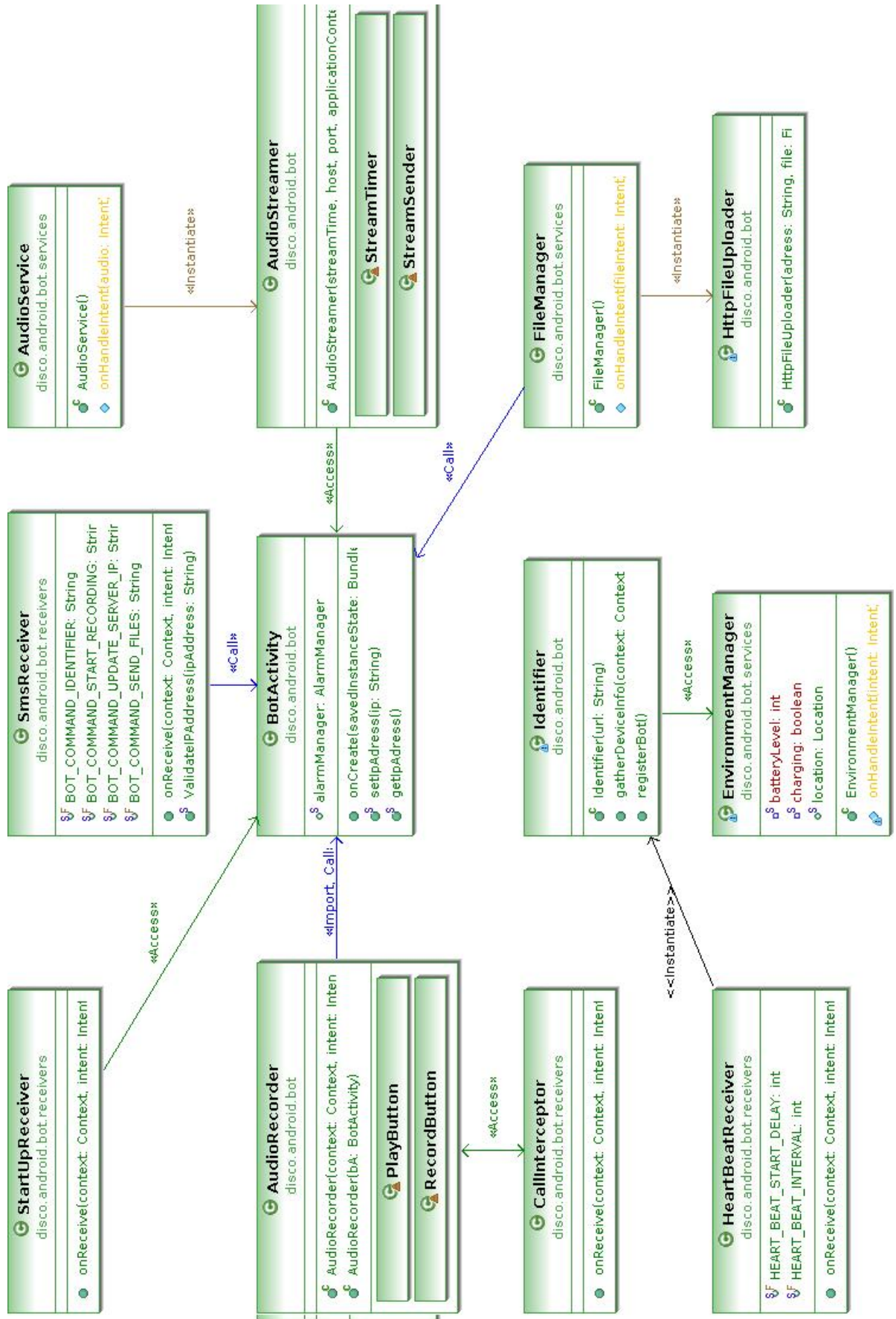


Figure 5.3: Simplified UML Model

5.4 Capabilities

5.4.1 Register and Home call

After installation, or after the device has booted, the bot gathers various information about the device including: phone number, IMEI (International Mobile Station Equipment Identity), IMSI (International Mobile Subscriber Identity), provider, model, sdk version and device location. This data is then periodically transmitted to the Botmaster via the Internet every hour like a system heartbeat. In the table of the Botmaster's webpage all zombie devices are listed, see chapter 6. For some providers, the phone number can not be extracted from the SIM card. In that case, the Botmaster has to wait until the bot sends a file of a recorded phone call. Then it can extract the phone's outgoing number which is included in the filename of a recorded phone call. Alternatively DiscoBot could send an SMS to a trusted phone, or directly to the Botmaster, but this probably involves additional costs.

5.4.2 Intercepting SMS

The SmsReceiver is the control unit of DiscoBot. Its responsibilities are to scan incoming SMS for the identifier, filter or rather delete them if the identifier is contained, and then to extract the command. The SmsReceiver extends an Android BroadcastReceiver and has a priority of 1000, which is the highest priority possible.

```
<receiver android:name=".receivers.SmsReceiver"
    android:exported="false"
    android:enabled="true">
    <intent-filter android:priority="1000">
        <category android:name="android.intent.category.DEFAULT"/>
        <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
    </intent-filter>
</receiver>
```

Figure 5.4: SmsReceiver As Declared In The Manifest

Every SMS_RECEIVED action triggers a *Broadcast Intent*, which means every broadcast receiver which has an *intent* filter for this action could potentially receive the SMS, but the SmsReceiver gets the SMS first, because its priority is higher than the system's. Android provides the method *abortBroadcast()* which prevents the SMS from being forwarded to a receiver with a lower priority.

As a result the SmsReceiver is able to receive and drop SMS which are intended for DiscoBot, that is, if a message starts with the string "BOT_COM:". Every incoming SMS is scanned for this identifier and will abort the Broadcast.

In conclusion, every SMS which starts with the bot command identifier, will be invisible to the rest of the system and the user. SMS without the identifier will be passed on to a receiver with the next highest priority.

After assuring that a message is intended for DiscoBot the SmsReceiver expects one of the following commands:

Command Identifier [optional parameter]	Effect
SRec[time in minutes]	starts to record audio for "time in minutes"/default time (1 min)
SRec[Stream,time in seconds[,hostname]]	starts to stream audio to "hostname"/default host
SeFi	forces to send recorded files to the server
USIp	the identifier must be directly followed by the new IP address

These are examples of commands, that DiscoBot understands right now, and other commands can easily be added.

5.4.3 Audio Recording

The AudioRecorder is responsible for all recordings. It both uses the built-in Android voice recognition parameter, and turns the speakerphone on to improve voice recording.

There are two different ways the AudioRecorder starts to record. One is via a SMS containing the SRec command, and the other is automatically via a phone call.

All recordings are stored in the application's internal storage space, which is inaccessible to the user, so that he or she can not accidentally find them. Filenames simply consist of a number which increases with every recording, unless it was a phone call. In that case the filename includes the incoming and the outgoing number. If the SmsReceiver receives a SMS with the SRec command the AudioRecorder is triggered using, if available, the transmitted time to record audio. Otherwise it records for a default time. Another option is to use the streaming parameter in the SMS to get real time audio transmission. Available options for the streaming mode are listed in the table in 5.4.2.

Every time the phone status changes the CallInterceptor is notified and, if the phone state changes to *off-hook*, the AudioRecorder is triggered and the call is being recorded no matter if it is an incoming or an outgoing call.

5.4.4 Environment Manager

The Environment Manager is periodically started and collects data about the current phone status containing: Wifi state, mobile state (Internet access *true* or *false*), percentage of used memory, battery level, battery charging (*true* or *false*) and the location.

The location is either a fine grained GPS location, or a more coarse grained network location. The Environment Manager tries to compute both for a maximum time span of 30 seconds. Whilst it is unlikely that the GPS location can be computed during such a short time, this is intentional in order to be stealthy, since GPS computation is one of the most power consuming functions. Neither the GPS nor the WLAN will be turned on in case they were initially off, so that it will not raise any suspicions. DiscoBot just hopes that the user is currently using GPS, WLAN or both, but does not take active measures, because a status change would be visible in the phone's status bar. The Environment Manager compares both locations and decides which one is better and if the the GPS location is not available, uses the last known one (if it is not older than 5 minutes) for comparison. Even if both WLAN and GPS are turned off DiscoBot still can compute the location using the network. However, this is the least favorable scenario, as it is the least precise method. The accuracy of the location variable varies thereby from only a few meters (GPS) to 1.3 kilometers (network), but if the phone is connected to a WLAN access point, which has a GPS location, the accuracy is denoted by 15-50 meter.

The collected data can then be evaluated and the Environment Manager decides if it is suitable to send the recorded data now, or to postpone this until the next scan. At the moment, the recorded files are sent to the server if the phone is both connected to a WLAN and the battery is charging. If this is the case, the Internet usage is free and DiscoBot does not need to be concerned about draining the battery. Thus it tries to send all files to the server. This is a very simple solution and since the Environment Manager provides additional information which could be taken into account to find a suitable moment, for example, that files are immediately sent if the memory is almost full. Another possibility for getting the recordings is to force DiscoBot to send the files with the "SeFi" command as described in the SmsReceiver [5.4.2].

6 The Botmaster

The Botmaster is the control unit of the Botnet and the only one who knows the whole dimension of the network. In this thesis the Botmaster is a simple web server connected with a *Nexus S* in order to send SMS commands. In practice it would also have a database to store all gathered recordings along with the history of every home call to create precise profiles of each zombie. A screenshot of the web interface is on the next page.

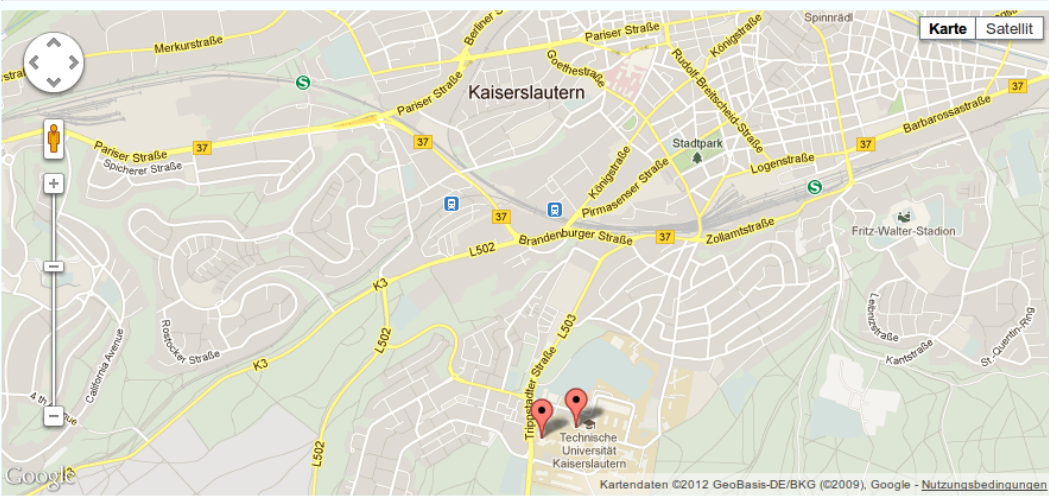
The table shows how all infected devices are listed with their respective information transmitted in the initial registration or in the last Heart Beat. Depending on the Heart Beat, the *last contact* entry is either green, yellow or red indicating if the device is calling in regularly. With a click on the *location* entry one can see the location of this specific device or with a click on *zombie devices*, as in the screenshot, the location of all devices. To send instructions to zombies, one or more *Nr.* entries can be selected, and after choosing a command the instructions will be sent via SMS using the connected phone. In the screenshot the Botmaster is about to send the SRec command to devices 2 and 3 without parameters, which could be added in the text field below.

Show entries Search:

Zombie Devices

Nr. ▲	IMEI	Phone#	IMSI	Model	Version	Registration	Last Contact	Location
1	000000000000000	15555215554	310260000000000	sdk	10	01/12/2011	20/11/2011 14:30:11	
2	355921041464189	01626480714	262026042783590	Nexus S	10	23/04/2012	23/04/2012 16:45:54	Kaiserslautern
3	355921041461821	01626480728	262026042783592	Nexus S	15	23/04/2012	23/04/2012 9:45:54	Kaiserslautern

Showing 1 to 3 of 3 entries ◀ ▶



Send Instructions

▼

Figure 6.1: The Web Interface

7 Evaluation And Prospect

7.1 Evaluation

In the following section the battery use of DiscoBot is evaluated. As mentioned before, low battery use is vital for the the stealthiness of the Bot. Below is a comparison of the battery drain with and without DiscoBot running.

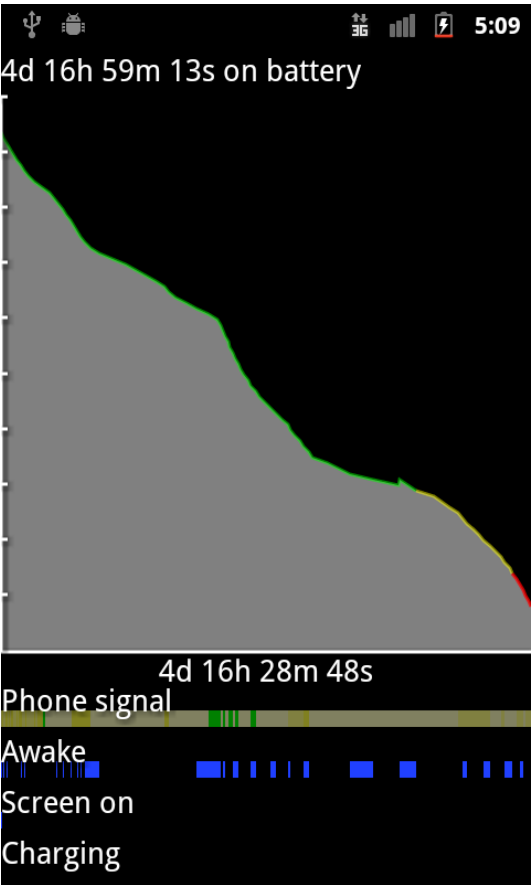


Figure 7.1: Battery Use With DiscoBot

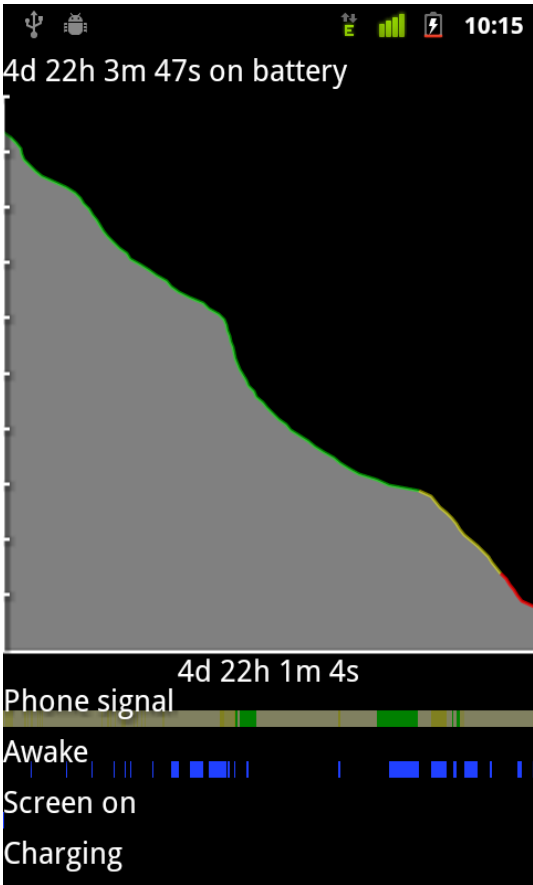


Figure 7.2: Battery Use Without DiscoBot

DiscoBot doing its hourly home call and position tracking shortened the battery life by only 5 hours and 4 minutes until the battery level dropped from 100% to 8%, which was decided as the cutoff point. During this comparison the phone was not used at

all, so the difference would shrink if the screen was on for a reasonable amount of time, or if other power consuming applications were used. In fact, both graphs appear to be virtually the same at first glance. Hence, it is extremely unlikely that the user would discover DiscoBot solely based on DiscoBot's battery use.

To evaluate the power consumption of DiscoBot's AudioRecorder, an example recording of one hour was recorded in a real life scenario. The resulting test file has 2,3 MB (3gp format) and consumed 8% battery life. Again, the screen was off and nothing else was in use during the recording. During the recording the phone lay on a desk and the file contains mostly background noise, which is why the file is so small, but every now and then snippets of a conversation can be heard.

7.2 Prospect

To improve the stealthiness of DiscoBot the functionality could be split into two separate applications, one to intercept SMS and the second to record and send audio. By using a shared UserID, see section 5.2.3, these applications would still be able to communicate between themselves, but the requested permissions would be split, raising fewer, or even no suspicions. A vast disadvantage of two separate applications is that both applications have to be installed for the bot to work properly.

Another possible improvement could be to preprocess the recorded audio files by deleting segments which contain only background noise. On the one hand, this would result in smaller files which would take less time to transmit, thereby potentially reducing cost, but on the other hand the power consumption would rise leaving it questionable if it would pay off.

8 Conclusion

Smartphone botnets are already a real threat and still continue to be so in the future, as this thesis shows. Their accompanied shortcomings are constantly diminishing as the devices themselves keep improving and communication costs decrease. With built-in sensors they even have some considerable advantages over computer botnets.

Until now the Android permission model is far from perfect as shown in section 5.2.2, but it is easy to improve, for example with finer grained permissions. DiscoBot is a good example of what can be done without exploiting the mobile phone, but rather exploiting a slightly gullible user, who would install the application.

The main advantage of a mobile botnet is the valuable information that can be gathered with a Smartphone. By using built-in sensors (GPS, microphone), or the video camera which almost every Smartphone has, it is easy to gain a wide range of sensitive information about the user. This thesis shows that this is already feasible and even in a very stealthy manner.

Android is still on the way up which is why malware for Android is rapidly increasing [12]. The average Smartphone users are not aware of the fact that their devices have basically the capabilities of a desktop computer, but come without a firewall or malware scanners. Such applications already exist, as described in chapter 3, and it is only a matter of time until they will be part of the basic system.

Ultimately it is the same never-ending battle as in the PC world between the bad guys, who are up to mischief, and the security researchers, who try to make the system secure.

Bibliography

- [1] Björn Marschollek André Egners and Ulrike Meyer. *Messing with Android's Permission Model*. 2012.
- [2] W.P.M. van Cuijk. *Enforcing a fine-grained network policy in Android*. 2011.
- [3] W. Enck, M. Ongtang, and P. McDaniel. "Understanding android security". In: *Security & Privacy, IEEE 7.1* (2009), pp. 50–57.
- [4] A.P. Fuchs, A. Chaudhuri, and J.S. Foster. "SCanDroid: Automated security certification of Android applications". In: *Manuscript, Univ. of Maryland*, <http://www.cs.umd.edu/~avik/projects/scandroidascaa> (2009).
- [5] Google. *Android Security Overview*. May 2012. URL: <http://source.Android.com/tech/security/index.html#Android-security-overview>.
- [6] Google. *Platform Versions Distribution*. May 2012. URL: <http://developer.android.com/resources/dashboard/platform-versions.html>.
- [7] Google. *The Android Permission Model*. May 2012. URL: <http://source.android.com/tech/security/index.html#the-Android-permission-model-accessing-protected-apis>.
- [8] Google. *What is Android?* May 2012. URL: <http://developer.Android.com/guide/basics/what-is-android.html>.
- [9] Adrian Kingsley-Hughes. *Android now at 850,000 activations per day*. Feb. 2012. URL: <http://www.zdnet.com/blog/hardware/android-now-at-850000-activations-per-day/18545?tag=content;siu-container>.
- [10] Adrian Kingsley-Hughes. *Millions caught up in Android botnet*. Jan. 2012. URL: <http://www.zdnet.com/blog/hardware/millions-caught-up-in-android-botnet/17891>.
- [11] N. Koblitz. "Elliptic curve cryptosystems". In: *Mathematics of computation* 48.177 (1987), pp. 203–209.
- [12] F-Secure Labs. *Mobile Threat Report*. Dec. 2011. URL: http://www.f-secure.com/weblog/archives/Mobile_Threat_Report_Q4_2011.pdf.
- [13] P. Maymounkov and D. Mazieres. "Kademlia: A peer-to-peer information system based on the xor metric". In: *Peer-to-Peer Systems* (2002), pp. 53–65.
- [14] C. Mulliner and C. Miller. "Fuzzing the phone in your phone". In: *Black Hat* (June 2009).

- [15] C. Mulliner and J.P. Seifert. "Rise of the iBots: Owning a telco network". In: *the Proceedings of the 5th IEEE International Conference on Malicious and Unwanted Software (Malware) Nancy, France*. 2010, pp. 19–20.
- [16] J. Oberheide and Z. Lanier. *Don't root robots!* 2011. URL: jon.oberheide.org/files/bsides11-dontrootrobots.pdf.
- [17] C. Papathanasiou and N.J. Percoco. *This is not the droid you're looking for...* 2010.
- [18] Margaret Rouse. *Exploit Definition*. Sept. 2005. URL: <http://searchsecurity.techtarget.com/definition/exploit>.
- [19] Margaret Rouse. *Rootkit Definition*. Jan. 2008. URL: <http://searchmidmarketsecurity.techtarget.com/definition/rootkit>.
- [20] R. Schlegel et al. "Soundcomber: A stealthy and context-aware sound trojan for smartphones". In: *Proceedings of the Network and Distributed System Security Symposium*. 2011.
- [21] A. Shabtai et al. "Google android: A comprehensive security assessment". In: *Security & Privacy, IEEE 8.2* (2010), pp. 35–44.
- [22] A. Shabtai et al. "Google Android: A state-of-the-art review of security mechanisms". In: *Arxiv preprint arXiv:0912.5101* (2009).
- [23] Techopedia. *What is a Botnet?* May 2012. URL: <http://www.techopedia.com/definition/384/botnet>.
- [24] P. Traynor et al. "On cellular botnets: Measuring the impact of malicious devices on a cellular network core". In: *Proceedings of the 16th ACM conference on Computer and communications security*. ACM. 2009, pp. 223–234.
- [25] Georgia Weidman. "Transparent Botnet Command and Control for Smartphones over SMS". In: *Shmoocon* (2011).
- [26] N. Xu et al. "Stealthy video capturer: a new video-based spyware in 3g smartphones". In: *Proceedings of the second ACM conference on Wireless network security*. ACM. 2009, pp. 69–78.
- [27] Y. Zeng, X. Hu, and K.G. Shin. "Design of SMS commanded-and-controlled and P2P-structured mobile botnet". In: *The University of Michigan, Ann Arbor, MI* (2010), pp. 48109–2121.