### Project Title:
Enhanced Network Security for Seamless Service Provisioning in the Smart Mobile Ecosystem

#### DELIVERABLE

<table>
<thead>
<tr>
<th>Deliverable No.</th>
<th>WP3</th>
<th>Task No.</th>
<th>T3.2</th>
<th>Lead Beneficiary</th>
<th>Dissemination Level</th>
<th>Nature of Deliverable</th>
<th>Delivery Date</th>
<th>Status</th>
<th>File name</th>
<th>Project Start Date</th>
<th>Project Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>D3.2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RE</td>
<td>P</td>
<td>31 July 2014</td>
<td>F: Final</td>
<td>NEMESYS_Deliverable_D3.2.pdf</td>
<td>01 November 2012</td>
<td>36 Months</td>
</tr>
</tbody>
</table>
### Authors List

<table>
<thead>
<tr>
<th>Author’s Name</th>
<th>Partner</th>
<th>E-mail Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laurent Delosières</td>
<td>Hispasec Sistemas</td>
<td><a href="mailto:ldelosieres@hispasec.com">ldelosieres@hispasec.com</a></td>
</tr>
<tr>
<td>Antonio Sanchez</td>
<td>Hispasec Sistemas</td>
<td><a href="mailto:asanchez@hispasec.com">asanchez@hispasec.com</a></td>
</tr>
</tbody>
</table>

### Reviewers List

<table>
<thead>
<tr>
<th>Reviewer’s Name</th>
<th>Partner</th>
<th>E-mail Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omer Abdelrahman</td>
<td>ICL</td>
<td><a href="mailto:o.abd06@imperial.ac.uk">o.abd06@imperial.ac.uk</a></td>
</tr>
<tr>
<td>Ravi Borgaonkar</td>
<td>TUB</td>
<td><a href="mailto:ravii@sec.t-labs.tu-berlin.de">ravii@sec.t-labs.tu-berlin.de</a></td>
</tr>
</tbody>
</table>
## Contents

**List of Figures**  
4

1 Introduction  
8

2 Attack vectors  
11

2.1 Attack vector exploiting vulnerabilities . . . . . . . . . . . . . . . . . . . 11

2.1.1 Metrics for the vulnerabilities . . . . . . . . . . . . . . . . . . . . . . . 11

2.1.2 Android vulnerabilities . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.2 Other attack vectors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

3 Crawler  
17

4 Client  
19

4.1 WebKit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

4.2 Android emulator . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

4.3 Androguard . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

4.4 Droidbox . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

4.5 Virtual user . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27

4.6 Cuckoo . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

5 Malware detector  
31

5.1 Dataset . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

5.2 Anomaly detection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

5.2.1 Machine learning algorithm . . . . . . . . . . . . . . . . . . . . . . . . 33

5.2.2 Feature vector . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36

5.3 Misuse detection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39

5.4 Performance evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44

6 Conclusions  
47
List of Figures

1.1 The honeyclient amid the NEMESYS architecture . . . . . . . . . . . . . 9
1.2 The honeyclient . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2.1 CVE details for the vulnerability 2010-1807 . . . . . . . . . . . . . . . 13
2.2 Android version share . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
2.3 Fake Google Play store “Blackmart” . . . . . . . . . . . . . . . . . . . . 15

4.1 The client . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
4.2 Script for launching an Android emulator . . . . . . . . . . . . . . . . . 22
4.3 Example of an Androguard log . . . . . . . . . . . . . . . . . . . . . . . 26
4.4 Example of a Droidbox log . . . . . . . . . . . . . . . . . . . . . . . . . . 28

5.1 Linear SVM in a two dimensional space . . . . . . . . . . . . . . . . . . . 34
5.2 Kernel mapping from input space to feature space . . . . . . . . . . . . . 35
5.3 Detection rate of each antivirus engine in VirusTotal . . . . . . . . . . . . 46
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADB</td>
<td>Android Debug Bridge</td>
</tr>
<tr>
<td>APK</td>
<td>Android Application File</td>
</tr>
<tr>
<td>CVE</td>
<td>Common Vulnerabilities and Exposures</td>
</tr>
<tr>
<td>CVSS</td>
<td>Common Vulnerability Scoring System</td>
</tr>
<tr>
<td>DCI</td>
<td>Data Collection Infrastructure</td>
</tr>
<tr>
<td>DEX</td>
<td>Dalvik EXecutable</td>
</tr>
<tr>
<td>ICCID</td>
<td>International Circuit Card ID</td>
</tr>
<tr>
<td>ICS</td>
<td>Ice Cream Sandwich</td>
</tr>
<tr>
<td>IMEI</td>
<td>International Mobile Equipment Identity</td>
</tr>
<tr>
<td>IMSI</td>
<td>International Mobile Subscriber Identity</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>MSRC</td>
<td>Microsoft Severity Rating System</td>
</tr>
<tr>
<td>NFC</td>
<td>Near Field Communication</td>
</tr>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>PoI</td>
<td>Point of Interest</td>
</tr>
<tr>
<td>QEMU</td>
<td>Quick EMUlator</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SMS</td>
<td>Short Message Service</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>XSS</td>
<td>Cross-scripting</td>
</tr>
</tbody>
</table>
Abstract

This deliverable presents the design of the honeyclient module of the NEMESYS data collection infrastructure, which is responsible for collecting and analyzing Android applications hosted on potentially malicious websites. The document provides a comprehensive description of the three components that comprise the honeyclient: (i) a crawler that builds the list of URLs to be visited, (ii) a client that visits the URLs and analyses freshly collected Android applications, and (iii) a malware detector that classifies the collected applications as malware or goodware. Each component is described at a conceptual level, focusing on its design and integration within the NEMESYS infrastructure.
1 Introduction

This document is the deliverable D3.2 resulting from the Task 3.2, "High-interaction honeyclient development", of the project NEMESYS. D3.2 provides the design of the Honeyclient which aims to collect new Android malware. The deliverables 1.1 and 2.1 motivate our choice for the study of the Android platform. Indeed, based on those two deliverables, Android is the most popular platform amongst mobile devices and the most targeted platform by malware.

According to K.Asrigo et al. [1], "honeypots are ephemeral machines created solely for the purpose of studying attacks on machines connected to the Internet, and destroyed soon after they are compromised to prevent an attacker from abusing system resources". They are waiting for clients to attack them. In other words, it consists in creating a system that can be attacked and monitored in order to learn the modus operandi of attackers. As opposed to honeypots, client honeypots or honeyclients are active in the sense that they actively search for malicious websites that are very likely to infect devices. Numerous Android-related papers have been published for the detection of Android malware such as [2, 3]. Honeypots for Android have also been proposed such as HoneypotLabSac [4], honeyM [5], HosTaGe [6], etc. for passively collecting Android applications. To the best of our knowledge, we have not encountered any honeyclients which were publicly available for collecting Android applications.

In the NEMESYS infrastructure, the honeyclient is part of the DCI as depicted in the Figure 1.1. The Data Collection Infrastructure is directly connected to the other components of the NEMESYS infrastructure, namely the honeypot on the Smart Mobile Device, the Access Network, and the Mobile Core Network. The honeyclient assists the other components of DCI in detecting anomalous traffic by collecting and classifying Android applications. It uses the external database of VirusTotal in order to improve its accuracy when classifying collected Android applications as malware or goodware.

The honeyclient consists of three main components, namely a crawler, a client, and a detector. The whole honeyclient is depicted in Figure 1.2. It first (1) receives a list of Points-of-Interests (PoIs) containing the list of web pages where it is very likely to be infected. This list is parsed by the crawler, described in Section 3, which crawls every PoI and their children, i.e., all the sub links. All the extracted links compose the set URLs (2) which is sent to the client described in Section 4. The client consists of two modules, namely a WebKit and an Android emulator along with the analyzers.
The former one will visit every URL, extract **APKs** (3), and send them to the Android emulator while the latter one will analyze the extracted **APKs** and will generate **logs** (4) out of the analyses. Those **logs** will be sent to the malware detector described in Section 5 to classify the **APKs** as **malware** or **goodware** (5) from the **logs**.

The rest of this deliverable is organized as follows. Section 2 presents a survey of existing attack vectors that allow the installation of malicious software on a mobile device, which provides a guideline for the design of the crawler and client components. The design of the crawler is presented in Section 3, while Sections 4 and 5 describe the client and malware detector, respectively.
Figure 1.2: The honeyclient
2 Attack vectors

This Section introduces the attack vectors that are commonly used to install malicious software on a mobile device. Listing the attack vectors will guide the design of the client. Indeed, facilitating the client’s infection will allow us to collect more Android applications. We will first introduce the Android vulnerabilities as a first attack vector using the exploit to automatically install malicious software and secondly we will see the other attack vectors. Finally, we will summarize the most used attack vectors. We first introduce the vulnerabilities of the Android platform as a first attack vector allowing automatic installation of malicious software, and then we discuss other attack vectors. The Section concludes with a summary of the most used attack vectors.

2.1 Attack vector exploiting vulnerabilities

A vulnerability typically arises from a weakness in a program, and can be exploited by an attacker in order to inject code remotely, gain root access, or perform other malicious activities without the user’s knowledge. In order to quantify the severity of a vulnerability, several metrics have been developed in the literature, which are discussed in the following.

2.1.1 Metrics for the vulnerabilities

The most used system to quantify vulnerabilities is Common Vulnerability Scoring System (CVSS) [7]. A CVSS, currently at the version 2, is defined by the following three metrics:

- Base: it represents the intrinsic characteristics of vulnerability, totally independent of time or environment

- Temporal: it defines the characteristics that can be changed over time.

- Environmental: it represents the characteristics that are relevant and unique for a specific user environment.
CVSS is not the only system for quantifying vulnerabilities. Indeed, other metrics are used such as CERT/CC [8] and Microsoft Severity Rating System (MSRC) [9]. CERT/CC was used by the CERT Coordination Center until March 2012. Since then, it is using CVSS. The metric is based on:

- The available information about the vulnerability.
- The exploitation of the vulnerability. Sometimes it is known that attackers using the vulnerability to attack systems.
- The risk to Internet infrastructure because of this vulnerability.
- The number of Internet-connected systems affected by this vulnerability.
- Once exploited the vulnerability, the impact that can be achieved with it.
- The ease of exploiting the vulnerability.
- The preconditions required to exploit the vulnerability.

CERT/CC scores a number between 0 and 180 that assigns an approximate severity to the vulnerability. The resulting score is not linear, i.e., a vulnerability with a score of 40 is not twice as severe as one with a score of 20.

The scoring system MSRC tries to provides insights into the severity of vulnerabilities affecting Microsoft’s customers using 4 levels:

- **Critical**: Used when the impact of the vulnerability is code execution without user interaction or without notice by the system.

- **Important**: This rating is used when the attacker can compromise the confidentiality, integrity or availability of the user or system resources. Also when the system is able to display warning or error screens while performing operation.

- **Moderate**: When the exploitation of the vulnerability depends on other factors such as user authentication settings that are not default.

- **Low**: When the impact of the vulnerability is mitigated by the default characteristics of the affected systems.
2.1.2 Android vulnerabilities

Android, like other systems, has vulnerabilities the most critical of which are the ones that allow an attacker to take control of the system, extract sensitive information, or cause a denial of service. In this study, we are only interested in the former one which allows an attacker to take the control over the device, install and execute malicious software remotely enabling us to collect the malicious software via the honeyclient.

The service CVE Details [10] is a free service that lists the security vulnerabilities of different systems. Amongst them, we can find the vulnerabilities that affect the Android platform [11]. This service displays the name of the vulnerability, the identifying number, the number of exploits, the vulnerability type, the date when the exploit has been discovered, the CVSS score, etc. An example of a vulnerability is depicted in Figure 2.1.

We have considered the vulnerabilities affecting the Android platform without the third-party software such as Adobe Reader, Java, etc. since they are not installed by default in most cases. At the time of surveying the vulnerabilities (January 2013), there have been 29 exploits reported for the Android platform [12] whose critical vulnerabilities. However, the affected Android versions represent a minority of Google devices: only 1% of the devices connected to any Google services. On the contrary, the versions between 2.3 and 4.1 represent an overwhelming majority of users with 89% of the market. The market share is depicted in Figure 2.2.

![Figure 2.1: CVE details for the vulnerability 2010-1807](image)

For instance, the vulnerability CVE-2010-1807 enabling to inject remote code only works for Android versions ranging from 1.0 to 2.1. It affects the WebKit rendering
engine used, among others, Apple Safari and Android’s default browser in its version prior to 2.2. This exploit allows a remote attacker to execute arbitrary code by opening a specially crafted HTML document.

![Android version share](image)

Figure 2.2: Android version share

### 2.2 Other attack vectors

Since the most used Android version, at the time of surveying the vulnerabilities, does not present any critical vulnerabilities publicly reported, we have considered other ways to infect mobiles. According to R.Unuchek [13] from Kaspersky Lab, there exits three main ways to distribute malware other than exploiting vulnerabilities:

- **Send bulk SMSs (spam campaigns) that contain a malicious link.** Upon clicking on the link, the user is asked to install a malicious application. Besides the SMS spam campaigns, there are also Email spam campaigns advertising Android malicious applications such as [14, 15].

- **Fake Google Play.** Attackers create a page that looks like the official Google marketplace and distribute fake URLs through internet. On the fake Google Play, we find applications that are usually not free. Usually, those applications have been repackaged to integrate malicious code. The user downloads and installs the malicious applications. An example of a fake Google Play, blackmart, is shown in Figure 2.3.

- **Compromise servers.** An attacker uses a persistent cross-scripting (XSS) vulnerability in a web page to inject a malicious JavaScript code. An example is provided
in Listing 2.1. Upon loading a web page, the javascript code embedded on the web page is interpreted by the Android browser. It checks the browser’s user agent and redirects the browser to a malicious application if the request comes from an Android browser, i.e., if the user agent contains the string "android" in the Listing 2.1.

```javascript
window.onload = function () {
    if (navigation.userAgent.match (/android/)) {
        window.location = 'http://domain.com/fraudulent.apk'
    }
}
```

Listing 2.1: Javascript redirection

Figure 2.3: Fake Google Play store "Blackmart"
2.3 Summary

No critical vulnerabilities, without taking into account the third-party software, were discovered for the most used Android versions, i.e., GingerBread and ICS for automatically installing malicious applications. Therefore, it is very unlikely that attackers will use Android platforms’ vulnerabilities to infect Android devices. Other attack vectors are used such as sending bulk SMSs with malicious links, faking Google Play, and injecting malicious code in compromised servers.

From the aforementioned information and the definition of the honeyclient, the fake Google Play and the compromised servers turn out to be the best sources to collect Android malware. However, we will only take into account fake Google Play that do not require any special API to collect Android applications. Indeed, we want our crawler to be generic for collecting Android application, i.e., to be the closest possible to users’ behavior browsing the web.
3 Crawler

In this Section, we shall see how we build the list of suspicious websites, i.e., the crawler’s algorithm.

We define a Points of Interest (PoI) as a resource shared by many people. A Point of Interest might be an online newspaper such as bbc.co.uk for instance. For getting malicious applications, we crawl web pages that are very likely to contain malicious code, i.e., PoI. According to Symantec [16], there are PoIs’ categories that are drawing the attention of attackers to infect devices and thus are very likely to contain malicious code. Symantec has ranked the most targeted infection sources for Personal Computers (PC) according to the percentage of total number of infected websites in 2011. We think that attackers will use the same infection sources to infect mobile phones. The ranking is as follows:

- Blogs (19.8%)
- Web hosting (15.6%)
- Business and economy (10%)
- Shopping (7.7%)
- Education and reference (6.9%)
- Technology, computer, Internet (6.9%)
- Entertainment and Music (3.8%)
- Automotive (3.8%)
- Health and Medicine (2.7%)
- Porn (2.4%)

Blogs turned out to be the first source of possible infection. This is due to the easiness of compromising a blog and injecting some malicious code. Furthermore, blogs are very often visited and constitute a very attractive infection source for attackers. Blogs using
frameworks such as Joomla [17], WordPress [18], etc. are very commonly exploited. Usually, the added plugins to the frameworks present vulnerabilities and are exploited to inject malicious code into a web server.

Based on the Symantec’s ranking, we have built a list of websites matching the highest targeted infection sources which constitutes our list of PoI. We use the Algorithm 1 to browse each PoI, extract links, and send the extracted URLs to the Client.

The Algorithm takes as arguments a URL to visit and the depth of visit. First, the URL is sent to the client via the function `SendURLToClient` that will visit it and extract eventual APKs. If the depth is different from 0, the links are extracted from the URL’s web page which constitute the children. Each child is itself visited by calling the recurrent function `VISIT`. The algorithm ends when all the children have been visited and their URL sent to the client.

Algorithm 1 Crawler’s algorithm

1: procedure VISIT(URL, depth)
2:    SendURLToClient (URL)
3:    if depth == 0 then
4:        return
5:    else
6:        extractedURLs = ExtractURLs(URL)
7:        for each extractedURL in extractedURLs do
8:            VISIT(extractedURL, depth - 1)
9:        end for
10:    end if
11: end procedure
4 Client

In this Section, we present the client, depicted in Figure 4.1, which collects and analyzes Android applications from the PoIs suggested by the crawler. Specifically, it consists of the following components: (i) a web browser (WebKit) for visiting the URLs given by the crawler, enabling to download Android applications; (ii) a hypervisor (Cuckoo) for dispatching the different Android applications to the analyzers, and managing the Android emulators (i.e., start and stop them); (iii) analyzers (Androguard and Droidbox) for performing static and dynamic analyses of the Android applications, and (iv) Android emulators for running the Android applications. Droidbox has been extended to integrate a Virtual User which instruments the Android applications while running in the Android emulators.

Figure 4.1: The client
4.1 WebKit

WebKit is a framework to render HTML, CSS, and JavaScript. It provides an API that allows to interact with it for visiting webpages, rendering webpages, and downloading webpages and files. The WebKit core is composed of three main components: an HTML syntactic analyzer, a rendering engine, and a JavaScript interpreter. Since 2005, the project is free and open software under the BSD license.

The information flow for WebKit is as follows: (1) WebKit receives URLs from the Crawler, (2) WebKit visits the URLs and waits for about 10 seconds, the time to render the webpages and get redirected to possible other webpages, (3) iterates over all downloaded resources, and (4) if any of these resources is an APK file, downloads and sends it to Cuckoo for its analysis. The rendering time was obtained through tests with fraudulent and legitimate URLs, and is the minimum time required to obtain all resources of a webpage taking into account possible webpage redirections.

In order to simulate a mobile device’s browser, we have replaced the user-agent of the WebKit with the Samsung Galaxy S2’s one. The user-agent is sent in each HTTP request enabling a server to identify the client’s browser. According to the user-agent, the server can propose different webpages. For instance, a server can propose a webpage that is customized for mobile devices if the client uses a mobile device’s browser. As seen in Section 2, attackers use the user-agent to distinguish between mobile devices’ browsers and computers’ browsers. When a mobile device’s browser is detected, a malicious Android APK is proposed instead of a webpage.

Moreover, using the WebKit instead of the Android emulator’s browser allows us to visit URLs faster since we do not need to start and stop the Android emulators. As a matter of fact, WebKit is more scalable.

4.2 Android emulator

We chose the Android ICS 4.1.1 as our Android emulator OS that allows to run the majority of Android applications from Gingerbread to ICS. However, if the application is only built for Android versions before ICS, this application will not be able to run with ICS. Nevertheless, it represents a minority of applications.

Each Android application is executed in an Android emulator. This allows to run it in a confined space and therefore prevent it from infecting the host machine. Each Android emulator is based on the Quick EMUlator (QEMU). To enable to restore the filesystem of the Android OS after executing an Android application, a clean filesystem’s snapshot of every Android emulator has been taken prior to every execution. Besides,
we also wipe out all the user data to remove any files stored by an Android application after each Android application execution. However, running the Android application in an emulator has some drawbacks such as execution speed which is slower since each instruction is interpreted by QEMU.

To communicate with the Android emulator, we use the Android Debug Bridge (ADB), which is a client-server program. When the client is running, it will communicate with the server running on the host machine. The server is responsible for establishing the communication with each Android emulator and keeps track of all the emulators that are switched on. ADB allows to transfer files, install Android applications, remove Android applications, inject events such as tap events, inject SMS, change geographical locations, etc.

For running the Android emulator, we used the bash script displayed in Figure 4.2. It calls the main program "emulator" built-in in the Android SDK for instantiating the emulator QEMU. It takes several parameters, namely in respective order the kernel, the Android firmware, the ramdisk.img containing extra drivers and programs used by the boot loader, the userdata.img which contains the new Android applications, the sysdir which specifies the system directory, i.e., the configuration of the emulator, the option "memory" for specifying the memory limit by the emulator, the option "no-window" for indicating the emulator to run as headless since we are running it on a headless server, the option "partition-size" for setting the partition size to 600MB, the option "no-boot-anim" for disabling the boot animation while booting the emulator, the option "prop" to set a property of the Dalvik VM, the option "tcpdump" to indicate the PCAP file in which the network traffic will be saved in, the options "snapstorage" and "no-snapshot-save" to tell the emulator to use the a snapshot image of the emulator while not saving its new state when switching off since we do not want to keep an infected Android emulator, and the verbose option to show the debugging information. We used the snapshot both for booting more rapidly the Android emulator and restoring the original state of the emulator before its infection. This process avoids us to overwrite the Android filesystem for restoring the Android emulator and therefore speeds up the analyses.

4.3 Androguard

Androguard provides a framework for statically analyzing Android applications. It enables to retrieve the Android permissions requested by an Android application, extract the list of called functions, re-assemble an application, detect the presence of ads and obfuscations making an analysis harder, etc. It also enables to disclose any dormant
codes that would not be executed by a dynamic analysis. Specifically, the following information can be extracted from an Android application:

- From the AndroidManifest file, which contains the configuration of the application, i.e., the activities, the services, the list of permissions that are required to access to the system resources, etc.:
  - The syntax of the AndroidManifest. There are some malware that malform the AndroidManifest to circumvent automated analyses. However, this malformed AndroidManifest is correctly accepted by Android smartphones.
  - The package name. It allows to identify the application on an Android smartphone. Every application must have a unique package name. In other words, no two applications having the same package name can be installed on the same smartphone.
  - The minimum version of Android SDK that is required to execute the Android application. If the Android smartphone is too old and therefore has an old SDK, the application cannot be installed on the smartphone.
  - The maximum version of Android SDK that is compatible with the Android application. If the Android smartphone is too recent and therefore has the most recent SDK, the application cannot be installed on the smartphone.
  - The list of permissions required by the application. When accessing to a

Figure 4.2: Script for launching an Android emulator
resource, the application must ask the authorization to the system. This is the enforced Mandatory Access Control.

- The target SDK. It specifies the SDK with which the application has been tested and does not require any extra work to maintain the forward-compatibility.

- The new permissions added. An Android application can propose new permissions. It is useful for instance when two applications developed by the same developer want to share resources. In this case, the application can create a permission and the other one can require it.

- The list of application’s activities. The activities represent the graphical interfaces of the application.

- The list of providers. Providers are interfaces to access and store data.

- The name of the main activity. The main activity is the activity that will be the first started.

- The list of receivers. Receivers register to events which enable them to execute some code upon an even triggering.

- The use of external libraries by the application.

- The list of libraries that the application must be linked to. For instance, the application might embed the google maps in its application and therefore rely on the google maps library. If the library is not installed, the application fails to install.

- The list of filters. Those filters enable to catch events. For instance, when the phone has booted, the event ”android.intent.action.BOOT_COMPLETED” is broadcasted.

- The list of services. Services in Android execute very long tasks in background.

- From the classes:

  - The number of classes composing the applications.

  - The average length of class names. Usually for making harder the reverse engineering, the class name are very short.

  - The average entropy of class names. For obfuscating the code and therefore hardening the reverse engineering of the code, the class names are chosen randomly, i.e., increasing the entropy.
– The presence of ascii obfuscations. It checks if the name of methods and classes is formatted in ASCII.

– The average number of goto per class. For obfuscating the code, it is very common to redirect the code very often making it harder to analyze it. This is achieved by using a lot of goto instructions.

– The total number of redirections in the application, i.e., goto.

– The presence of functions for accessing to the browser history.

– The presence of functions for accessing to the sensors. By sensors, we intend the accelerometer, the GPS, etc.

– The name of the sample.

– The presence of functions for getting the SIM serial number or ICCID.

– The presence of functions for getting the phone number.

– The presence of functions for getting the GPS location.

– The presence of functions for getting information on the different accounts. An account might be a Google account, Microsoft account, etc.

– The presence of functions for handling the Near Field Communication (NFC).

– The presence of functions for getting the device id (IMEI).

– The presence of functions used by Ads. We detect the main ads, namely Mobclix [22], AdMod [23], Airpush [24], and LeadBolt [25].

– The presence of functions for getting the serial number.

– The presence of functions for getting the list of installed applications.

– The presence of functions for accessing to the Camera.

– The presence of functions for running native binaries. By native binaries, we intend binaries that were written in C or C++ and are compiled for a certain family of processors. Usually, it is very uncommon to encounter this case since most of the interesting feature are present in the SDK.

– The presence of function for accessing to the subscriber id (IMSI).

– The presence of functions for sending SMS.

– The presence of functions for sending MMS.

– The presence of functions for sending data.

– The presence of functions for sending ciphered data.
- The presence of functions for dynamically registering to an event.
- The presence of the reflection code. In Java, it is possible to dynamically load a class via the reflection method.
- The presence of phone calls functions.
- The presence of Mobile Country Code (MCC) functions.
- The presence of dynamic code.
- The presence of cryptographic functions.

- From the certificate:
  - The content of the certificate. This includes the name of the issuer, the subject, etc.
  - The certificate signature.

### 4.4 Droidbox

Droidbox [26] automates the analysis of an Android application. When executed, the Android application calls Android functions that are provided by the Android framework. A subset of those functions are hooked by DroidBox and output a log when executed. This subset encompasses the functions that send out data, read from or write into files, use cryptographic functions, load classes, make a call to a phone, send SMS, etc. The output log is retrieved by means of the tool Android Debug Bridge (ADB) [27] and then parsed in order to extract the logs generated by DroidBox. DroidBox also includes the module TaintDroid [28] for tracking the leakage of sensitive data.

TaintDroid uses a modified version of an Android firmware for tracking the sensitive data. It modifies the core function of the Android firmware in order to tag the sensitive data, transfer the tags when a variable is copied, and sum up tags when two variables with two different tags are concatenated for instance. This is done both at the Dalvik VM and Java frameworks level. All the functions that access to a sensitive information are modified in order to tag the variable containing the sensitive information. So as to detect the leakage, the functions that send out data are also modified to check if a tagged variable is about to be sent out. If so, a log is generated. In other words, every sensitive information that is sent out (e.g., a phone number in a SMS) is logged. We have considered as sensitive data: the phone contacts of the phone book, the phone number, the GPS locations, the SMS, the IMEI, the IMSI, the ICCID, the device serial number, the accounts (e.g., Google account), and the browser history.
For hooking the functions, DroidBox w.r.t to its version provides two ways: the former one consists in modifying the Android firmware, i.e., modifying and recompiling the Android firmware while the latter one consists in modifying the Android applications in order to add the hooks inside this application. However like malware for Desktop PCs, Android malware can use techniques for hiding the functions and load them dynamically and therefore bypass the Droidbox hooking. As for the second one, a malware
can bypass the hooking at the Dalvik level by instantiating a C program which interacts directly with the native functions. However, for calling most of the native functions, the user must have the root access which is not set by default when an application is launched. As for the applications that use elevation of privileges, they represent a minority and generally exploit vulnerabilities of old Android platforms. However, since we have chosen the Android emulator ICS, we limit the circumventions. All the characteristics that are dynamic are preceded by a timestamped indicating at what time, the characteristic has been used. The following characteristics are recorded:

- **apkName**: the filename.
- **dataleaks**: the sensitive data that were leaked through SMS, files, and network.
- **opennet**: the communications that were initiated.
- **recvnet**: the network traffic that was received.
- **sendnet**: the network traffic that was sent.
- **servicestart**: the services that were started.
- **sendsms**: the SMS that were sent.
- **cryptousage**: the cryptographic functions that were used with the key.
- **accessedfiles**: the files that were accessed, i.e., read and written.
- **dexclass**: the DEX files that were loaded.
- **hash**: the application’s hashes.
- **phonecalls**: the phone calls that were made.

### 4.5 Virtual user

In order to get an analysis as complete as possible of the malware, the client is instrumented by a framework integrating a virtual user simulator and a user actions recorder [30]. The simulator is built from a model which is based on eight different reproducible scenarios composed of about five user actions each. An action is a click on a button, a swipe, etc. We chose the application Facebook, Hotmail, Youtube, Calendar, Gallery ICS, Browser, Slide Box Puz, and talk.to since they encompass most of the
user actions and are amongst the most downloaded. All those applications run on the Android ICS emulator. Before recording the user actions, the Android is first set up, i.e., an account is created for the applications Facebook, Hotmail, and talk.to.

The following scenarios have been constructed for the applications:

- **Facebook**: the user (1) signs in, (2) posts a new message, (3) goes to the list of messages by clicking on the top icon Messages, and then (4) opens the first message.
- **Hotmail**: the user (1) signs in, (2) clicks on the first message, (3) scrolls down, and then (4) presses the button "Home" to go back to the Android board.
- **Youtube**: the user (1) scrolls down the list of videos, (2) plays one video, (3) increases the volume, (4) decreases the volume, (5) goes back to the list of videos by clicking on the button "Back", (6) scrolls up the list, and then (7) plays another video.
- **Calendar**: the user (1) signs in (2) selects one day with a long pressing, (3) types the name of an event, (4) clicks on Save, and then (5) goes back to the Android board by clicking on the button "Home". (6) signs out.

---

Figure 4.4: Example of a Droidbox log

```json
{
    "apkName": "binaries/7748
c43994e5c076651902fbc9d3bf975ccf6cac4c4214a4c60d376c830dc171",
    "dataleaks": {},
    "opennet": {},
    "recvnet": {},
    "sendnet": {},
    "servicestart": {"0.8888800144195557": {"type": "service", "name": "com.android.battery.BridgeProvider"}},
    "sendsms": {},
    "cryptousage": {},
    "accessedfiles": "{1856937235": "2f70726f632f3736352f636d646c696e65"},
    "dexclass": "{0.7522540092468262": "/data/app/identity.android.dbCounter-1.apk", "type": "dexload"},
    "hash": ["3118b5b2c5a7f84d9b9d21293eae53", "e854ebd42822b4d33f7409aadbe2f5e03dfdd066", "7748
c43994e5c076651902fbc9d3bf975ccf6cac4c4214a4c60d376c830dc171"],
    "phonecalls": {}
}
```
• Gallery ICS: the user (1) selects one album, (2) selects one photo, (3) zooms in, and then (4) zooms out.

• Browser: the user (1) selects main menu in Android, (2) clicks on the browser icon, (3) selects keyboard, and then (4) types ”www.peugeot.com”.

• Slide Box Puz: the user (1) moves the ball left, right, up, and down for the 1st level.

• talk.to: the user (1) selects a contact, (2) types a message, (3) presses the button Settings, and then (4) presses the item End chat.

After playing each scenario, the main activity is brought to the front. Indeed, when playing a scenario, depending on the design of the tested application, we can exit the application at the end of the scenario after pressing the button ”Back”. For this, we use the ADB command to force the tested application to be in the foreground.

Since future Android malware might embed virtual machines detectors by checking the GPS locations or track the accelerometers changes for instance, in addition to the previous scenarios, we inject some random events so as to make the phone look more real to the Android applications. For instance, events of the GPS locations are injected into the Android emulator at the middle of the analysis time.

This virtual user has been integrated inside DroidBox so as to instrument the tested application when DroidBox starts it. Before the first analysis, DroidBox loads the different scenarios into memory. This allows to access to the events composing the scenarios in a faster way while reducing the access to the hard drive.

4.6 Cuckoo

Cuckoo [31] is a framework for automating analyses of Windows binaries and visits of URLs. It is composed of a machine manager, scheduler, and analyzer. The machine manager allows to manage a virtual machine, i.e., starting and stopping a virtual machine. Since there exits many different types of virtual machine, Cuckoo has a plugin for each of them (e.g., VirtualBox, VMware, etc.). The scheduler looks after dispatching the tasks to the different available virtual machines and getting the logs at the end of each analysis. As for the analyzer, it is responsible for analyzing the state of the Virtual Machine, e.g., tracking the network traffic and extracting the HTTP connections, taking memory snapshots, trace hooks, etc.

For the communication between the host and the guest, Cuckoo uses a client/server model. For this, it integrates a client module in the guest which communicates with the
server of the host for fetching URLs to browse or application to install, and to give back the logs. This model enables Cuckoo to work in a distributed environment and therefore enables to be scalable. It also interfaces a REST interface allowing remote users to fetch the logs and sending other URLs or Windows binaries to analyze.

Since Android emulators were not integrated into Cuckoo, we have modified Cuckoo. The modification includes the adding of a plugin for managing the Android emulator, the modification of the scheduler to interact with the framework Droidbox and Androguard and retrieve their analysis. The Android emulators are managed by means of the program ADB which allows us to track the state of each Android emulator. However, for transferring the URLs or Android applications to analyze, we directly invoked DroidBox and Androguard and then retrieved their analysis to store them in Cuckoo. This first version is a prototype which could be easily distributed later on by considering DroidBox as a client.
5 Malware detector

The malware detector consists of an anomaly detector that is able to identify new malware families and a misuse detector to recognize already known malware. It classifies an Android application as malicious if any of the two detectors consider it so. The use of two types of detectors is intended to improve the overall performance of the system, which ideally should exhibit a high detection rate and a very low false positive rate. In particular, detection rate corresponds to the rate of malware that have been classified correctly, while false positive rate refers to the rate of goodware that have been misclassified as malware.

The rest of this Section is organised as follows. First the dataset used for training and testing the malware detector is presented. Then we describe the design of the anomaly and misuse detectors. Finally, we evaluate the performance of the two detectors.

5.1 Dataset

For building and training the malware detector, we needed both malware and goodware datasets. For starting up, about 2000 malware and 2700 goodware have been collected. More datasets are expected to be available during the course of the project.

The goodware dataset consists of the top 100 free applications of the 27 categories of GooglePlay. Google Play encompasses 27 different categories: game, books and reference, business, comics, communication, education, entertainment, finance, health and fitness, libraries and demo, lifestyle, app wallpaper, media and video, medical, music and audio, news and magazines, personalization, photography, productivity, shopping, social, sports, tools, transportation, travel and local, weather, and app widgets.

For automatically downloading the applications from GooglePlay, a google account has been created with a Samsung Galaxy SII associated. Without the association of the phone, it is not possible to access to the applications. One reason is that some applications are customized for a certain type of hardware such as those using native libraries or a particular version of the android SDK. Also, external libraries are compiled for a given processor architecture. For downloading the applications automatically, we have used the python framework “android-market-api-py” [32].

Google Play uses the service, Bouncer, for detecting malware before they get uploaded.
However, as presented by C.Miller et al. [33], it is easy to circumvent Bouncer and upload malware to the Google Play. For instance, the malware can activate its malicious code after the time out of Bouncer, or exploit the vulnerabilities of QEMU on which Android emulator is based.

In order to reduce the possible number of malware from Google Play, all the Google Play applications have been sent and analyzed by VirusTotal except for those exceeding the file size restriction of 64MB. 10% of the goodware were detected as adware by at least one antivirus engine, i.e., displaying ads and sending sensitive information to a remote server (e.g., phone ID). The detected adware represent the most common adware such as LeadBolt [25] and AirPush [24]. Nevertheless, we also encountered some Planktons and variants of Plankton, first detected in Google Play in 2011 [34], which are considered as more virulent unlike the other adware. They send the Android version, the IMEI, entice the user to download another application by redirecting him to the Google Play, etc. A sample was classified as goodware if it was not detected by any antivirus engines. Therefore we have discarded all the Plankton, LeadBolt, and Airpush.

The malware dataset is composed of malware downloaded from VirusTotal and from third party web sites. A report of each malware by VirusTotal was retrieved in order to know the detection rate. A sample was classified as malware if it was detected by at least 20 antivirus engines. This threshold enables us to discard any false positives.

All the malware and goodware, after passing through VirusTotal, have been fed into Cuckoo of the honeyclient in order to get both a static and dynamic analysis. However, some applications could not execute properly because the AndroidManifest was corrupted and therefore it was not possible to get the main activity to start, or there were no main activities to start, etc. Only the malware and goodware correctly executed have been kept for the malware detector while the other ones have been discarded. In the end, 1943 goodware and 1363 malware have been analyzed correctly.

All the samples have been analyzed during 6 minutes and instrumented by the six scenario during the analysis as described in Section 4. Besides the scenarios, events such as the geographic localization and a SMS have been sent to the emulator in order to look more realistic. Geographic localization simulates a person moving while executing the application. Those events have been sent to the emulator in order to trigger any dormant code of the application that are triggered by the events. An event might be a SMS that is received for instance. It is the case of the Zbot Mobile version known as Zitmo [35] which steals SMS received from a bank and upload it to the attacker either by SMS or network connection.
5.2 Anomaly detection

An anomaly detector works by finding patterns in data that do not conform with expected behavior. For training the anomaly detector, there exists two main approaches: supervised learning, and unsupervised learning. The former learning consists in feeding a machine learning algorithm with a labelled dataset composed of both goodware and malware samples. During the testing phase, any unseen data is compared against the two models to determine its class, i.e., goodware or malware. A supervised learning anomaly detector is also called classifier. As for unsupervised learning, the machine is trained by an unlabelled dataset based on a hypothesis of an overwhelming majority of normal data over the abnormal data in the dataset. The characteristics of the dataset, i.e., the presence of both malware and goodware labels, allow us to use the supervised learning approach.

5.2.1 Machine learning algorithm

We have chosen the Support Vector Machine (SVM) [36] as the machine learning. It has been introduced in the mid-1990s as a classifier. In SVM, the problem has been formulated as having a clear separation between two classes, goodware and malware in our case, which are separated by a hyperplane. The two margins at the hyperplane sides enable to keep the data points as far as possible from the hyperplane. In other words, finding a clear separation consists in finding the best hyperplane maximizing the distance between the hyperplane and the two margins at its sides, i.e., pushing away the data points from the hyperplane. In Figure 5.1, we have represented two classes whose data points are two-dimensional: one with the data points in white and the other one in black. The hyperplane is the line while the two margin limits are delimited by two dashed lines. The hyperplane represents the separation between the two classes. The points crossing the dashed lines are called the support vectors because they support the two margins.

The hyperplane is modelled by the equation \( w \cdot x + b = 0 \) with its two margins modelled by the respective equations \( w \cdot x + b = 1 \) and \( w \cdot x + b = -1 \). \( w \) is the normal to the hyperplane, \( x \) represents the feature vector of an Android application in our case, and \( b \) is a real number. Since the distance between the margin with the hyperplane is \( \frac{1}{\|w\|^2} \), maximizing the margin distance from the hyperplane comes to maximizing \( \frac{2}{\|w\|^2} \). As the dual problem, this comes to minimizing \( \frac{1}{2}\|w\|^2 \). Besides the minimization, we need to model the fact that each data point needs to be located in its correct class, i.e., malware data points must belong to the malware class and vice versa. This constitutes the constraints and can be modelled as follows \( (w \cdot x_i + b) y_i \geq 1 \).
we need to model the fact that each data point needs to be located in its correct class, i.e., malware data points must belong to the malware class and vice versa. This constitutes the constraints and can be modelled as follows \((w . x_i + b).y_i \geq 1\) where \(y_i\) representing the label of \(x_i\) and belongs to the set \([-1, +1]\). \(-1\) represents the malware class while \(+1\) represents the goodware class.

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} ||w||^2 \\
\text{subject to} & \quad (w^T . x_i + b).y_i \geq 1, \forall i \in \{1, \ldots, N\}
\end{align*}
\]
An other approach has been proposed to relax the problem by introducing soft-margins where we tolerate errors. We note $\xi_i$ the error with the cost $C$ induced by each error. The problem remains the same, i.e., maximizing the distance between the margins and the hyperplane while keeping the error as low as possible. This problem is modelled as follows:

$$\minimize_{(w \in \mathbb{R}^d, b \in \mathbb{R}, \xi_i \in \mathbb{R}^+)} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{N} \xi_i$$

subject to $$(w^T x_i + b) y_i \geq (1 - \xi_i) , \xi_i \geq 0 , \forall i \in \{1, \ldots, N\}$$

Figure 5.2: Kernel mapping from input space to feature space

In the previous problems, the hyperplane was linear in the input space but it might turn out that it is not the case and therefore we need to map the data points from the input space into the feature space in order to be able to find a linear hyperplane in the feature space. For this, we map the data points into the feature vector with a kernel function that maintains the scalar product, i.e., $x_i \cdot x_j = \Phi(x_i) \cdot \Phi(x_j)$ where $\Phi$ is the kernel function that maps the data points from the input space to the feature space. Figure 5.2 shows an example of a mapping from the input to the feature space and the deformation of the hyperplane modelled in red. In the feature space, the problem can be modelled as follows:
minimize \( \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{N} \xi_i \)

subject to \((w^T \Phi(x_i) + b)_i y_i \geq (1 - \xi_i)\), \(\xi_i \geq 0\), \(\forall i \in \{1, \ldots, N\}\)

Several kernel functions exist that map the input space to the feature space. Amongst them, we have the most common:

- The polynomial kernel \(\Phi(x_i, x_j) = (x_i \cdot x_j)^d\)
- The polynomial inhomogeneous \(\Phi(x_i, x_j) = (x_i \cdot x_j + 1)^d\)
- The gaussian radial basis function \(\Phi(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)\)
- The hyperbolic tangent \(\Phi(x_i, x_j) = \tanh(k x_i \cdot x_j + c)\)

We use the function \(f(x) = sgn(w^T \Phi(x) + b)\) to evaluate the class of a feature vector \(x\) where the function \(sgn\) gives +1 if its input is positive and -1 otherwise. An output of +1 means that the feature vector has been classified as goodware and malware otherwise.

**5.2.2 Feature vector**

Before training and testing the anomaly detector, we need to build the feature vector out of the DroidBox and Androguard analyses. In other words, we need to select the relevant features that best characterize the goodware and the malware. If we keep all the features, we get very poor performance of the anomaly detector. This problem is also known as overfitting where the built models tend to overfit the goodware and malware models and are not realistic. To select the relevant features, there exist three main methods:

- **Filter methods** [37] are preprocessing methods that use statistical tests to assess the feature relevance. This is usually done before applying any classification algorithms. They use the statistical properties of the features to filter out the irrelevant features and usually order the features or nested subsets of features. Principal Component Analysis (PCA) and Fisher Criterion Score are some examples. They are really robust against over-fitting but may fail to select the most "useful" features.

- **Wrapper methods** [37] offer more accurate results than filter methods but are more computationally demanding. They create all possible feature subsets which turns
out to be computationally expensive and use the cross-validation to assess the subset feature relevance. In principle, they can find the most "useful" features but they are prone to over-fitting.

- Embedded methods [37] are very similar to Wrapper methods. But unlike wrappers, the possible feature subsets are generated during the machine learning process which is less computationally expensive. For assessing the feature subset relevance, they use cross-validation. They are similar to wrappers but are less computationally expensive and less over-fitting.

To select the relevant features, we have used a filter method proposed by YW. Chein et al. [38] which uses a Support Vector Machine with a gaussian radial basis function and the F-Score metric. F-Score is a simple technique measuring the discrimination of two sets. Given the training vectors \( x_k \) with \( k = 1, \ldots, N \), if the number of malware and goodware instances are \( n_+ \) and \( n_- \), respectively, the F-Score of the \( i \)th feature is defined as follows:

\[
F(i) = \frac{(\bar{x}_i^+ - \bar{x}_i)^2 + (\bar{x}_i^- - \bar{x}_i)^2}{\frac{1}{n_+-1} \sum_{k=1}^{n_+} (x_{k,i}^+ - \bar{x}_i^+)^2 + \frac{1}{n_-+1} \sum_{k=1}^{n_-} (x_{k,i}^- - \bar{x}_i^-)^2}
\]

where \( \bar{x}_i, \bar{x}_i^+, \bar{x}_i^- \) are the average of the \( i \)th feature of the whole, malware, and goodware data sets, respectively. \( x_{k,i}^+ \) is the \( i \)th feature of the \( k \)th malware instance while \( x_{k,i}^- \) is the \( i \)th feature of the \( k \)th goodware instance. The numerator represents the discrimination between the malware and the goodware sets while the denominator represents the discrimination within each of the two sets. The larger the F-Score is, the more likely this feature is more discriminative.

The algorithm proposed by YW. Chein et al. [38] is as follows:

1. Select the training set \( X_{training} \) containing malware and goodware’s raw feature vectors with their respective label. The raw feature vector of an instance is the concatenation of the droidbox and androguard logs. Since the SVM machine cannot handle non floating features, all the non floating features have been mapped to a floating number.

2. Shuffle randomly the set \( X_{training} \).

3. Compute the F-Score value for each feature and order the features by their F-Score value ranging from the highest value to the lowest value.
4. Compute the largest features subset to keep. We only keep the features whose F-Score value is greater than a certain threshold which has been fixed to $10^{-20}$.

5. Form the possible feature subsets to try. Start with the largest features subset. Decrease the subset size by 2 until it remains only two elements in the set. At each step, a new possible feature subset is formed.

6. Evaluate all the features subsets previously formed on the machine learning. Perform a cross validation on the $X_{training}$ set to determine the optimal SVM parameters. It means that within the $X_{training}$ set, one part will be used for training the machine learning while the rest will be used for testing the machine. For this, we use the algorithm developed by YW. Chein et al. [38].

7. Select the best subset features, i.e., the subset that shows the best accuracy rate, i.e., showing the best detection rate and the lowest false positive rate.

The feature selection algorithm was optimizing the parameters $C$ and $\gamma$ of the SVM in order to get the best accuracy for each feature subset. It is worth mentioning that those parameters will be fixed for the rest of the deliverable. To fix the parameters such as the cost $C$ and the gaussian radial basis function parameter $\gamma$, YW. Chein et al. [38] uses the following algorithm:

1. Consider a grid space of $(C, \gamma)$ with $\log_2 C \in \{-5, -3, \ldots, 15\}$ and $\log_2 \gamma \in \{-15, -13, \ldots, 3\}$
2. For each hyperparameter $(C, \gamma)$ in the search space, conduct 5-fold cross validation on the training set.
3. Choose the parameter $(C, \gamma)$ that leads to the best cross-validation’s accuracy rate.

Before applying the feature selection algorithm, we have selected the features which best characterize both malware and goodware. We call raw feature vector an input vector resulting of the concatenation of Androguard and Droidbox logs where all the non integer values have been mapped to integer values. The feature selection algorithm has been applied on the raw input vector of the dataset containing 600 samples with 300 goodware and 300 malware. All the raw feature vectors that have been processed by the feature selection algorithm are referred as feature vector. The following features have been chosen:

- **SMS_functions**
• average_length_class_names
• average_entropy_class_names
• IMSI_functions

The feature SMS_functions characterizes Android malware that use premium SMS which represent an overwhelming majority of Android malware. The features average_length_class_names and average_entropy_class_names characterize Android malware that use obfuscation. Obfuscation is a technical employed for hardening reverse engineering and might be achieved by shortening class names and using a low Shannon entropy of class names. As for the last feature, it enables to get the phone ID.

We can notice that all the selected feature result from the static analysis. We were expecting it since the dormant code in Android malware is not as easy to execute even tough every Android application was instrumented by the client. Moreover, we suppose that during the analysis, the C&C were down preventing the malware to reveal its complete malicious behavior because the samples were one year-old.

5.3 Misuse detection

A misuse detector consists in matching existing patterns that belong to malware. Since the service VirusTotal [39] encompasses 48 different misuse detectors engines which combined together offer the best misuse detector of the world, we have elected it as our misuse detector. It keeps up to date the latest malware signatures. We will use the VirusTotal private API to send a malware and get its analysis. Nevertheless, VirusTotal presents drawbacks such as the file size limit to 64 MB that can be submitted to VirusTotal and the frequency of accessing to the antivirus analysis which is fixed to 2 requests per minute. However, it is entirely sufficient for our Proof-of-Concept but it could be increased later on if necessary.

Like anomaly detectors, misuse detectors also have false positives. In order to reduce the false positive rate, we have fixed a threshold upon the number of detection engines. We have estimated that by fixing the threshold to 20 antivirus engines, about half of the antivirus that are able to detect Android malware. The list of antivirus currently integrated in VirusTotal is as follows:

• Agnitum. This antivirus engine is offered by the russian company Agnitum Ltd created in 1991. Originally the company focused on anti-trojan and PC connections monitoring solutions. Later on, they have widened their services by developing personal firewall and Internet security products.
• AhnLab-V3. The antivirus engine is developed by AhnLab which is a security company from South Korea founded in 1995. Besides offering antivirus products, they also provide online security, network security appliances such as firewalls, etc.

• AntiVir. This is a German antivirus software product created in 1988. Avira is the sixth largest antivirus vendor worldwide.

• Antiy-AVL. It is an antivirus engine developed by the Chinese company Antiy. The company has been founded in 2000.

• Avast. This is a Czech antivirus engine created in 1988. Avast is the acronym of "Anti-Virus - Advanced Set". Avast! Free Mobile Security reached best-rated security application in Google Play.

• AVG. This is also a Czech antivirus engine created in 1992. In first instance, they were only proposing an antivirus solution. Then they have extended it to anti-spyware.

• Baidu-International. Baidu international antivirus engine innovated original ultrafast cloud security technology. Baidu Antivirus utilizes cloud computing technology and its massive file database to quickly and accurately eradicate the latest trojans, unknown trojans, and other malicious programs. This solves the problems faced by traditional antivirus software such as the lag behind the latest trojans and viruses and the huge consumption of computer resources.

• Bkav. It is antivirus engine developed by Vietnam Cyber Security Center Bkis. It was first built in 1995.

• BitDefender. It is antivirus engine developed by the Romanian software company Softwin. It was first launched in 2001. The software provides an antivirus engine, anti-spyware, firewall, e-mail spam filtering, backup, tune-up, and parental control components.

• ByteHero. This antivirus engine has been developed by the Chinese company ByteHero. Its engine detection is based on a dynamic and static code analysis. It is based on new detection technology, does not include virus database and is capable of detecting new unknown viruses on the internet without being upgraded.

• CAT-QuickHeal. It is proposed by an Indian company.

• ClamAV. This antivirus engine is free and open source.
• Commtouch. This antivirus is developed by the Israelite company founded in 1991. It also includes an anti-spam, URL filtering, mail reputation services, zombie intelligence, spam protection, pattern detection, and IP reputation.

• Comodo. Developed by the company Como Group founded in 1998, it offers antivirus and firewall protection products.

• DrWeb. It is developed by a Russian IT security. It became the first antivirus service in Russia in 1992.

• Emsisoft. It is an antivirus and antispyware protection suite developed by Austria-based Emsisoft GmbH. It integrates a Dual-engine scanner (BitDefender and Emsisoft’s own Anti-Malware scanner) with more than 13 million signatures (January 2013), a Behavior blocker which is able to detect unknown zero-day attacks without signatures, and a Surf protection that blocks malicious hosts when trying to access them.

• ESET-NOD32. Made by a Slovak company, it was first created in 1987. It contains both an anti-malware and anti-spam module.

• F-Prot. It is being developed by the Icelandic software company FRISK founded in 1993. They built an antivirus and anti-spam service.

• F-Secure. This antivirus is built by the Finish company F-Secure Corporation, formerly known as Data Fellows. It was founded in 1988.

• Fortinet. Fortinet is an American company specialized in network security appliances. It was founded in 2000.

• GData. It is a German antivirus software that was first created in 1985. It is one of the eldest security software companies in the world. They also have a URL scanner to detect malicious websites.

• Ikarus. It has been created by the Austrian security company IKARUS Security Software in 1986. It is a pioneer of the software industry focusing AntiVirus and content security.

• Jiangmin. This is a Chinese antivirus product from the company Beijing Jiangminxinke Technology established in 1996. They provide a network Viruses Software Single, a firewall for mail servers, an antivirus software and a series of information security products.
• K7AntiVirus. It has been developed by the company K7 Computing Pvt which provides a solution for malware prevention and antivirus software. It uses signatures and heuristics algorithms to identify malware.

• K7GW. He founded K7, India’s first and exclusive antivirus software company in 1991. It offers antivirus software, firewall, anti-spam, antispyware, adware blocker and privacy control service, etc.

• Kaspersky. Kaspersky Lab is a Russian multi-national computer security company founded in 1997. Kaspersky Lab ranks fourth in the global ranking of antivirus vendors. It is a developer of secure content and threat management systems.

• Kingsoft. This is one of China’s oldest software companies. It was founded in 1989 as a company focused on developing internet and Microsoft windows-version systems. They released their first antivirus in 2000. Kingsoft Internet Security 9 Plus is an anti-virus and security application designed for Internet users. It contains antivirus, anti-malware, a vulnerability scanner and personal firewall.

• Malwarebytes. Made by Malwarebytes Corporation, it was first released in January 2008. It scans for and removes malware when started manually, and a paid version, which additionally provides scheduled scans, real-time protection and a flash memory scanner.

• McAfee and McAfee GW Edition. Founded by an American company in 1987, the company first proposes an antivirus. Later on, they have widened their services by offering anti-spyware software, data loss prevention, etc.

• Microsoft. Microsoft has released the Microsoft Security Essentials in 2009 incorporating antivirus, anti-spyware, etc. It provides real-time protection for the computer equipped with Windows XP, 7, and 8.

• MicroWorld-eScan. Escan is an antivirus developed by the american company MicroWorld technology. Its first version was released in 1993. Escan also provides anti-spam and content security solutions which include protections against malware, spyware, etc.

• NANO-Antivirus. It is a Russia’s anti-virus software which is designed to protect your computer from viruses, trojans, worms and other malicious software. It is developed by "NANO Security". Their first issue was released in 2009.
• Norman. Founded in 1984, the company from Norway is very active in the field of data security. They offer products such as antivirus, personal firewall, anti-spam, encryption, and parental control.

• nProtect. It is a Korean antivirus company founded in 2000 also known as INCA Internet Corporation. The antivirus engine is composed of two engines: Tachyon (engine developed by the company), and BitDefender engine.

• Panda. The antivirus has been developed by the Spanish company Panda founded in 1990. They were the first to propose a Panda Cloud Antivirus which protects computer from the Panda cloud. The files are scanned in the Cloud without using the processor unit’s user machine. Panda Cloud Antivirus is able to detect viruses, trojans, worms, spyware, dialers, hacking tools, jokes and other security risks.

• PCTools. The original PC Tools package was first developed as a suite of utilities for DOS, released for retail in 1986. The company was founded in 2003.

• Rising. It is a Chinese software company that produces the anti-virus software Rising Antivirus, a firewall, UTM and spam-blocking products. The company was founded in 1991. It is the China’s largest antivirus software company.

• Sophos. Sophos began producing its first antivirus and encryption products in 1985. It detects and eliminates viruses, worms and Trojans on your computer or network.

• SUPERAntiSpyware. It is a software application distributed as shareware which can detect and remove spyware, adware, trojan horses, rogue security software, computer worms, rootkits, parasites and other potentially harmful software applications. Even though it can detect malware, it does not replace an antivirus.

• Symantec. Symantec corporation is a global computer security software corporation. It was founded in early 1982. They are famous for the Norin Antivirus first released in 1990.

• TheHacker. It has been created by the Peruvian company HackSoft. The first version was released in 1992.

• TotalDefense. This antivirus is issued from the merge of the antiviruses CA Anti-Virus and VET Anti-Virus.

• **VBA32.** It detects and neutralizes computer viruses, computer worms, Trojan horses and other malware (backdoors, adware, spyware, etc.) in real time and on demand. It has been developed by the Belarus company VirusBlokAda.

• **VIPRE.** It is antivirus created by Sunbelt Software company. Acquired by GFI Software company, its products include software for filtering spam and viruses from e-mail as well as for monitoring and scanning networks for security purposes.

• **ViRobot.** It is antivirus developed by the Latin American company Hauri.

An antivirus is able to analyze any file upon their content. It scans the file, extracts any statical information and compares it to its virus database’s signatures. Upon matching, a file is considered as a malware. A signature is a set of bytes at given locations that characterize a malware family. An antivirus can detect Android malware if the corresponding signature is present in the database. However, currently, antiviruses tend to incorporate a heuristic engine which allow them to detect new malware’ variants. For instance, in the Kaspersky engine [40], the heuristic engine counts the number of suspicious commands. If this counter exceeds a certain threshold, the application is considered to be probably malicious. Antiviruses also embed a dynamic heuristic engine where the application is executed in a secure environment, more specifically a part of the program is emulated. If any suspicious actions are detected while emulating the execution, the application is also considered to be malicious.

### 5.4 Performance evaluation

After selecting the features, all the raw feature vectors were transformed into feature vectors. For the anomaly detector, we kept the SVM built by YW. Chein et al. [38] with the optimal performance ($C$ to 256 and $\gamma$ to 0.03125). For evaluating the anomaly detector, we have used a cross-validation which consists in dividing a dataset into two-folds: one fold of the set is used for learning the machine while the second fold is used for testing the machine learning. The training set was composed of 300 malware and 300 goodware and the rest of the two sets was composing the testing set, i.e, over 80% of the two sets. In order to know the unbiased performance of the anomaly detector, we have tested it on the same set randomized 200 times. On the average, we get a detection rate of 90.5% and false positive rate of 7.4%. In the worst case, we get a detection rate of 86.2% and a false positive rate of 10.1%. In the best case, we get a detection rate of 93.6% and a false positive rate of 5.2%. It is worth noting that the performance does not represent the malware detector performance since we have omitted the misuse detector. With the misuse detector, we expect to get a higher detection rate.
In order to quantify the number of antiviruses that can potentially detect Android malware, we have empirically assessed it by taking randomly 1000 Android malware downloaded from VirusTotal in November 2012. We hypothesize that since those malware are one-year old, the signatures must have already been created. Out of 1000 Android malware, in September 2013, only 5 antivirus engines have not detected any of the Android samples, namely nProtect, ByteHero, Malwarebytes, TheHacker, and SUPERAntiSpyware as shown in Figure 5.3. In other words, only 41 antivirus are able to detect Android malware.

Like Anomaly detectors, antiviruses suffer from false positive rate. In order to reduce the false positive rate, we will consider that an Android application is a malware if it has been detected as such by at least half of the antiviruses that are able to detect antiviruses, i.e., 20 antiviruses.
Figure 5.3: Detection rate of each antivirus engine in VirusTotal
6 Conclusions

In this deliverable, we have presented the design of the honeyclient that actively crawls the web to search for malicious Android applications. It is composed of a crawler that builds the list of URLs to visit, a client that visits the URLs and analyzes freshly collected Android applications, and a malware detector that classifies the collected Android applications as malware or goodware. We have surveyed the different vulnerabilities present in each Android OS as well as the Android OS versions the most used without taking into account the third-party software. We concluded that at the time of the study the most used versions (Gingerbread and Jelly Bean) do not present any critical vulnerabilities that would enable an attacker to inject remote code, i.e., an attacker would not be able to install automatically a malware. Therefore, we elected an emulated version of the Android browser (WebKit) for visiting the URLs and collecting Android applications. The emulated version runs faster than the Android emulator since we do not need to start and stop the Android emulator for visiting each URL. As a matter of fact, we can visit more URLs. As for the Android applications’ analysis, we chose the Android emulator Jelly Bean since we can run a wide range of Android applications starting from version Android 2.1 to the latest version (Jelly Bean). To analyze an Android application, we have used a static analyzer (Androguard) and dynamic analyzer (Droidbox), enabling to get the characteristics of the application. After applying the feature selections algorithm, we realized that only four static features (SMS_functions, average_length_class_names, average_entropy_class_names, and IMSI_functions) were relevant for distinguishing goodware from malware, and therefore every feature vector is composed of those four features. Finally, the malware detector comprising both a misuse detector (VirusTotal) and an anomaly detector (SVM), allows us to achieve 7.4% false alarm rate on average and at least 90.5% detection accuracy on average, for a dataset of 2000 malware and 2700 goodware.


48
Bibliography


Bibliography


