Vaccinating APK's

Milan Gabor
Viris
Ljubljana, Slovenia
milan@viris.si

Danijel Grah
Viris
Ljubljana, Slovenia
danijel.grah@viris.si

***Abstract*—this paper describes a tool that was developed for the purpose of dynamical analysis of Android applications. The tool is named Vaccine and can be used for browsing objects, fields and methods that the application uses at runtime. With the Vaccine it is possible to modify objects and fields and also execute code in the context of the running application at runtime.**

***Index Terms*—Android, Vaccine, dynamical analysis, android applications. penetration testing, APK,**

I. Introduction

The growing number of Android based devices, the simplified development process of Android applications and their wide spread usage is attracting potential attackers that are after financial gain.

Everything started when we were visiting conference Hack in Paris [1], where two guys impressed us with some interface to APK instrumentation.

Our work is addressing security issues of Android applications (APK's). We were facing quite some number of requests for security evaluation of mobile applications on Android. Analyzing this area we found out that there is no such thing as good tool to help with runtime analysis and we are too lazy to debug all the time. Therefore we developed a tool that is called the Vaccine and is used for dynamically analyzing APK's.

II. Dynamical analysis

We refer to dynamical analysis as a process of steps that are accomplished at runtime of an Android application and include accessing and changing the state of the application.

III. Vaccine

The Vaccine is a tool that prepares the Android application, installs it on mobile device or emulator, connects to the application and after that displays an user interface (UI), which is the entry point for dynamical analysis.

A. Preparation and installation of an APK

To maximize the gain of dynamical analysis in terms of getting knowledge about the structure and logic behind the Android application, the Vaccine is making some changes in the application. The whole process of preparation and installation can be stated as:

- unzipping the APK,
- baksmaling classes.dex,
- injecting service and libraries,
- smaling source to classes.dex,
- modifying AndoidManifest.xml,
- replacing classes.dex and AndroidManifest.xml in the APK,
- removing the signature,
- signing the application,
- installing the application,
- running the injected service,
- connecting and running the UI.

B. Architecture

The Vaccine consists of two parts. One part is the modified Android application; the other part is running on the computer. The parts are communicating throw TCP with the help of ADB. Below is a picture of the Vaccine architecture.

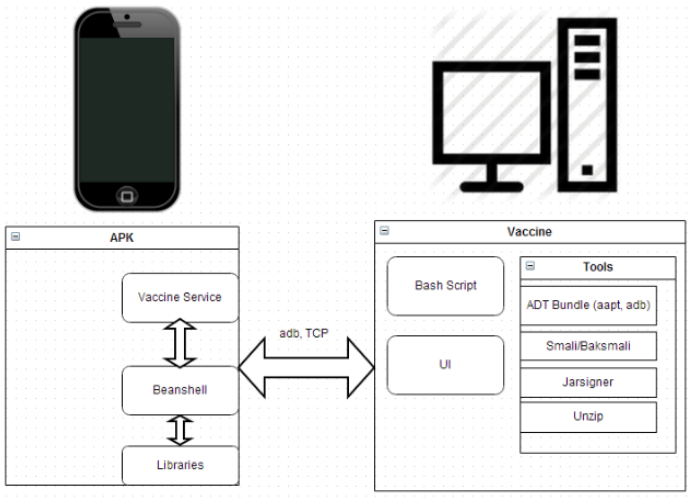

Figure 1: Architecture of the Vaccine

We have already described the process of preparing the APK. This is done by a Bash Script that uses tools shown in Figure 1. The final and modified APK has an injected service that is started after the installation. Besides that, the APK also embeds Beanshell. *BeanShell is a small, free, embeddable Java source interpreter with object scripting language features, written in Java [2].* Because Beanshell interpreter uses Java

reflection, it is suitable for Java source code execution at runtime on Android powered devices.

The entry point for dynamically analysis is the UI of the Vaccine. Figure 2 shows the Vaccine UI.

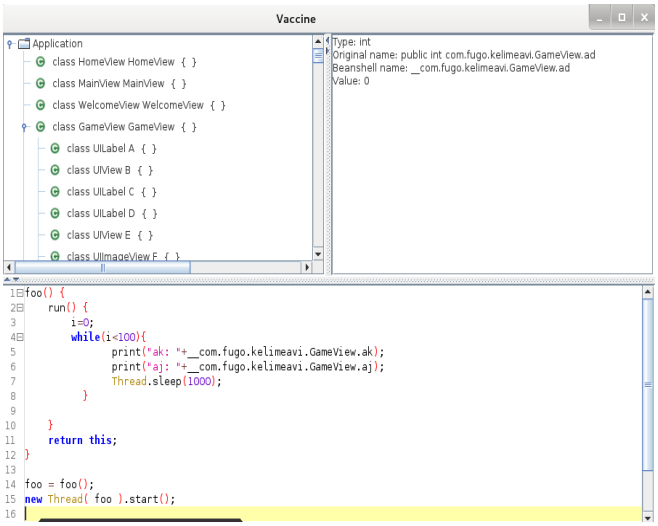

Figure 2: Vaccine UI

We divided the UI interface into three parts. The first part in the left upper corner of the Vaccine UI is a tree structure of objects that the application uses at runtime. The root object named Application was created artificial and represents the analyzing application. One level below are shown activities and the application context of the modified application. Activities are added to the root object automatically according to the activity life cycle management in Android [3]. The right upper corner of the Vaccine UI is a status field which displays some information about an object selected in objects tree. The third part of the Vaccine UI is a scripting area (below part one and two). There is possible to write Beanshell scripts as also strict Java source code. Scripts will be executed at runtime. There are some limitations. It is possible to create new objects from defined classes in the application or from classes in the Android API, but defining new classes it's not possible at the moment. Figure 2 shows a script where we created a scripted object "foo". This object has a method "run" that writes values of two variables to Android standard output. All is executed inside a new created Thread.

C. Dynamical Analysis with the Vaccine

To start analyzing an APK with the Vaccine, first the APK needs to be downloaded from the Android device. This can be done with the device connected to the computer and with help of ADB command

```
adb pull /full/path/to/apk. (1)
```

Then the Vaccine Bash Script is run like

```
vaccine –i apk_file –p 8888 (2)
```

After option "i" we give the Vaccine the Android application APK file, that we want to analyze. Next to option "p" is the port that the Vaccine service is listening on. After executing command 2, the Vaccine prepares and modifies the APK, install's it and runs the Vaccine service. Then it connects to the service and pops up the Vaccine UI. Now we can run the application on the device. After right clicking the root object in the tree object structure on the UI, we noticed that activates are added. The tree object structure responses to three actions: double click, left and right mouse click. With the left mouse click we are selecting the object and some information are displayed in the status field of the Vaccine UI. With the right mouse click on the object we are expanding the object. This means that the Vaccine returns all fields and methods of an object whether they are private or public and attaches them to the clicked object. When double clicking an object the Beanshell name of it is copied to the scripting part of the Vaccine UI. The Beanshell name is the name by which the object is accessible with the Vaccine. This name is a variable that can be used in our scripts.

D. Possibilites

Because the Vaccine uses smali and baksmali for compiling and decompiling the classes.dex file it is possible to analyze a large set of APK's. The Vaccine possibilities are limited to the device and permissions of an APK on that device. On a smart phone with a custom ROM like CynogenMod that is rooted by default and with access to system certificate it was possible to access the running Phone instance. We accomplished this by creating a new application, signing it with the system certificate and setting in the AndroidManifest.xml the attribute `sharedUserid` (in the manifest element) to `android.uid.system` and the attribute process (in the application element) to `com.android.phone`. Section of code shown below presents how to get the running phone instance with the Vaccine.

```
import com.android.internal.telephony.*;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import android.telephony.SmsMessage;
import android.app.PendingIntent;

Phone phone=null;
__beanshellNameOfActivity.runOnUiThread (
       new Thread(new Runnable() {
              public void run() {
                     phone =
        PhoneFactory.getDefaultPhone();
                     }}));
```

By adding the code shown below that was found at [4] and also simplified it is possible to send SMS messages with class 0.

```
IccSmsInterfaceManager          ismsm          =
       phone.getIccSmsInterfaceManager();
```

```
Field f =
       IccSmsInterfaceManager.class.getDeclared
Field("mDispatcher");

f.setAccessible(true);
SMSDispatcher sms_disp =
              (SMSDispatcher)f.get(ismsm);

byte[] b = new byte[0];
SmsMessage.SubmitPdu pdus =
              SmsMessage.getSubmitPdu(null,
"Phone number", "Zero SMS", false);
size = (int)pdus.encodedMessage[2];
size = (size/2) + (size%2);

pdus.encodedMessage[size+5] = (byte)0xF0;
sms_disp.sendRawPdu(pdus.encodedScAddress,pdus.
encodedMessage,null,null,"Phone number");
```

When injecting the Vaccine Service into useful APK's such as Mms.apk in CynogenMode the modified application can be used as a framework. For example it is possible to send SMS with a Beanshell script.

E. Reusability and extensions

There is work still going on using Beanshell commands and there is a lot of possibilities for further research and development of this or similar tools.

IV. Conclusions

We created the Vaccine, a tool for dynamical analysis of Android applications. It can be used for runtime code execution in the context of the running application. Therefore the Vaccine is suitable for testing how security is implemented in APK's. By importing classes from the Android API it is possible to use the Android device as a framework for creating simple scripts and testing the security model of Android.

References

[1] (2013) https://www.hackinparis.com

[2] (2013) What is Beanshell? Accessible on: http://www.beanshell.org/intro.html

[3] (2013) Activity. Accessible on: http://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle

[4] (2013) Zero SMS. Accessible on: https://github.com/virtualabs/ZeroSMS/blob/master/project/src/com/android/zerosms/ZeroSMS.java