Android Reverse Engineering & Defenses

Bluebox Labs

Patrick Schulz, Felix Matenaar

23./24. May 2013

BLUEBOX

Who we are

Patrick Schulz & Felix Matenaar

- Sr. Developer and Researcher
- working for Bluebox Security

- Mobile Enterprise Data Security Startup
- Stealth mode

BLUEBOX

A motivating example

RE: The Developer's perspective

Imagine you would develop an Android application.

RE: The Developer's perspective

Imagine you would develop an Android application.
The App includes...

- Fancy tricks and patterns
- Algorithms which have cost you lots of resources to develop
- Knowledge gained from company internal research projects

BLUEBOX

RE: The Developer's perspective

Imagine you would develop an Android application.
The App includes...

- Fancy tricks and patterns
- Algorithms which have cost you lots of resources to develop
- Knowledge gained from company internal research projects

Then you release through an official market,
and people start looking into your app...

Static Information Gathering

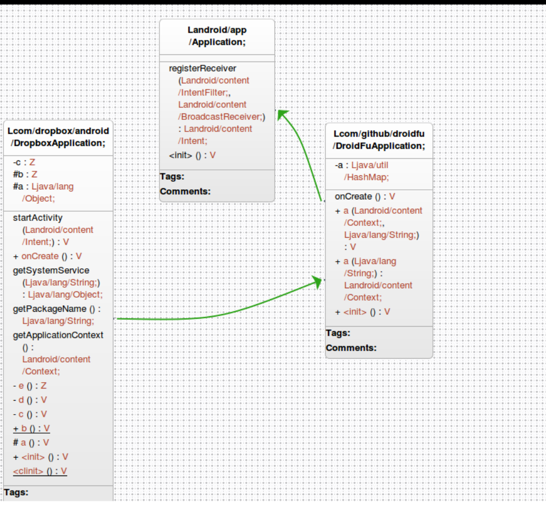
Decompilation

```
private void d()
{
    java.lang.String s = getPackageName();
    if(s == null || !s.startsWith("com."))
        s = "com.dropbox.android";
    android.content.Intent intent = new Intent("android.intent.action.VIEW", android.net.Uri.parse((new StringBuilder()).append("market://details?id=").append(s).toString()));
    intent.setFlags(0x10000000);
    if(com.dropbox.android.util.a.a(this, intent))
    {
        android.content.Intent intent1 = new Intent(this, com/dropbox/android/activity/UpgradeMessageActivity);
        intent1.setFlags(0x10000000);
        startActivity(intent);
        startActivity(intent1);
        android.os.Process.killProcess(android.os.Process.myPid());
    }
}

private boolean e()
{
    int i;
    java.util.Iterator iterator;
    java.util.List list = ((android.app.ActivityManager)getSystemService("activity")).getRunningAppProcesses();
    i = android.os.Process.myPid();
    iterator = list.iterator();
_L4:
    if(!iterator.hasNext()) goto _L2; else goto _L1
_L1:
    android.app.ActivityManager.RunningAppProcessInfo runningappprocessinfo = (android.app.ActivityManager.RunningAppProcessInfo)iterator.next();
    if(runningappprocessinfo.pid != i || !runningappprocessinfo.processName.equals("com.dropbox.android:message")) goto _L4; else goto _L3
_L3:
    boolean flag = true;
_L6:
    return flag;
_L2:
    flag = false;
    if(true) goto _L6; else goto _L5
_L5:
}

protected final void a()
{
    java.lang.Object obj = a;
    obj;
    JVM INSTR monitorenter ;
    b = true;
    a.notifyAll();
    return;
    throw <no variable>;
}
```

http://dexter.bluebox.com

BLUEBOX

Examining the Type System



http://dexter.bluebox.com

BLUEBOX

Examining APIs

https://www.dropbox.com/privacy?cl=%s&mobile=1
https://www.dropbox.com/help/category/Mobile?cl=%s#category:Mobile
https://www.dropbox.com/c/help/mobile_favorites?cl=%s&device=android
https://www.dropbox.com/android_opensource?cl=%s&mobile=1
http://www.example.com/example
https://www.dropbox.com/c/help/camera_upload_full?cl=%s&device=android
https://www.dropbox.com/terms?cl=%s&mobile=1
market://details?id=
http://
content://com.dropbox.android.provider.SDK
content://com.dropbox.android.LocalFile
file:///android_asset/js/pw.html

http://dexter.bluebox.com

Dynamic Analysis

Dynamic Analysis

- Some Sandbox implementations out there

  `http://www.honeynet.org/node/783`

- APKTool uses DDMS to debug disassembled Android applications

  `http://code.google.com/p/android-apktool/wiki/SmaliDebugging`

- Locate
  - licensing checks
  - data validation
  - client-side security
  - ...

BLUEBOX

App Modification

Application Modification

```
.-(~/Downloads/apktool1.5.2)----------------------------------------------
`--> ls
./  ../  apktool.jar  someapp.apk
.-(~/Downloads/apktool1.5.2)----------------------------------------------
`--> java -jar apktool.jar d someapp.apk
I: Baksmaling...
I: Loading resource table...
I: Loaded.
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/felix/apktool/framework/1.apk
I: Loaded.
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Done.
I: Copying assets and libs...
.-(~/Downloads/apktool1.5.2)----------------------------------------------
`--> ls -la someapp
total 40K
drwxrwxr-x  4 felix felix 4.0K May 14 14:06 ./
drwxrwxr-x  3 felix felix 4.0K May 14 14:06 ../
-rw-rw-r--  1 felix felix  18K May 14 14:06 AndroidManifest.xml
-rw-rw-r--  1 felix felix  213 May 14 14:06 apktool.yml
drwxrwxr-x 48 felix felix 4.0K May 14 14:06 res/
drwxrwxr-x  5 felix felix 4.0K May 14 14:06 smali/
```

BLUEBOX

Application Modification

```
.method public onCreate()V
    .locals 13

    .prologue
    const/16 v9, 0x114

    const/4 v12, 0x3

    const/4 v11, 0x2

    const/4 v2, 0x0

    const/4 v1, 0x1

+   invoke-static {}, SOMECLASS->someEvilMethod()V

    sget v4, Lcom/whatsapp/DialogToastListActivity;->f:I
```

BLUEBOX

Application Modification

```
.-(~/Downloads/apktool1.5.2/someapp)------------------------
`--> ls
./  ../  AndroidManifest.xml  apktool.yml  res/  smali/
.-(~/Downloads/apktool1.5.2/someapp)------------------------
`--> java -jar ../apktool.jar b
I: Checking whether sources has changed...
I: Smaling...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
.-(~/Downloads/apktool1.5.2/someapp)------------------------
`--> ls dist
./  ../  someapp.apk
.-(~/Downloads/apktool1.5.2/someapp)------------------------
`-->
```

Consequences

Consequences

- Trivial to reverse engineer Android applications
  - Static Analysis supported by available metadata
  - Dynamic Analysis using debugging or sandboxes
- Easy to repack applications
  - Add Malware to a benign application
  - Circumvent licensing checks

Agenda

How can we address these problems?

- Static Analysis
  - Identifier & Code Obfuscation (not in this talk)
  - Callgraph obfuscation
  - Dynamic code loading
- App Modification
  - Manifest cheating
  - Runtime integrity checks
- Dynamic Analysis
  - Debugger detection
  - Debugger prevention

Anti-Static Analysis

Anti-Static Analysis

- Plenty of analysis tools available

  http://resources.infosecinstitute.com/android-malware-analysis/
- Manual analysis
  - make code harder to read
  - crash analysis tool
  - fool analysis tool
- Automated analysis
  - crash analysis tool
  - fool analysis tool

Callgraph obfuscation

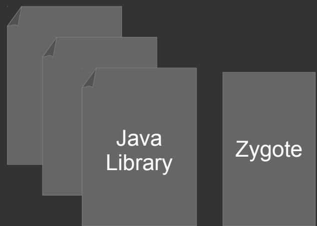
- An app starts with a fork of zygote process
- Has preloaded lib as well as the Android framework
- Include classes in your APK which are defined in preloaded libs

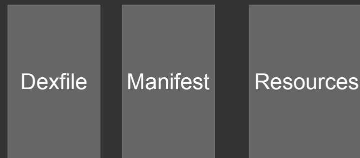
- Bytecode points to the APK internal definition
- During runtime the preloaded definition will be used

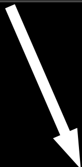
Example:

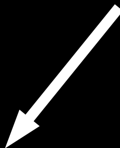
android.os.Process

BLUEBOX


Zygote Process
Java Library
Zygote

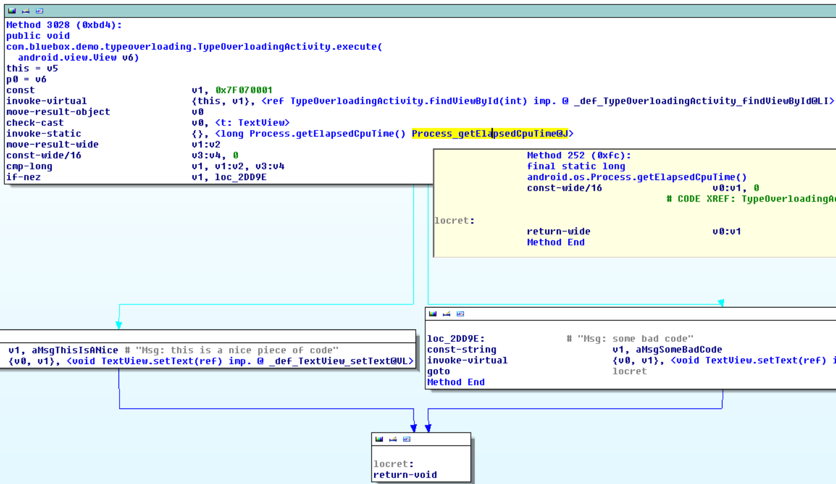

Android Application (.apk)
Dexfile
Manifest
Resources



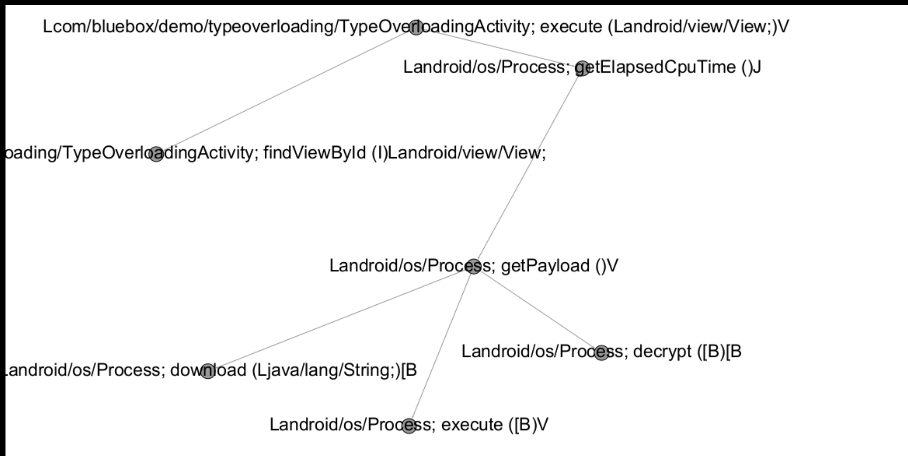


Running Application
Java Library
Dexfile

Callgraph obfuscation - IDA Pro



Callgraph obfuscation - androguard



BLUEBOX

Callgraph obfuscation - Fix

- Know the execution environment
- Filter for classes that are preloaded
- Keep an eye on vendor specific preloaded libraries

BLUEBOX

Dynamic Bytecode Loading

- Static analysis can only consider statically available bytecode
- Further bytecode can be loaded during runtime
  - using classloader
  - using "class DexFile"
  - using native builtin Dalvik functionality

Dynamic Bytecode Loading

- Bytecode distribution:
  - Encrypted in the APK, e.g shipped as asset or resource
  - Downloaded during runtime
- Makes static analysis very expensive
- Ask dynamic guys for help ;)

Tamper Proofing

BLUEBOX

Tamper Proofing

- React to the fact that your app has been modified or repacked
- Hide interesting code paths from dynamic analysis
- Make it harder for malware authors to repack your app

✓ ARM's Trustzone

✓ Signed by vendor to get system level access

× Not applicable for the usual app developer

Manifest cheating

- AndroidManifest.xml included in APK file
- Purpose: Define application meta data
  - Requested permissions
  - Registered components like services and activities
- Represented using binary format in APK

```
1 <application android:name="optional.entry.class">
2     <activity android:name="com.example.manifestexample.MainActivity">
3         <intent-filter>
4             <action android:name="android.intent.action.MAIN" />
5         </intent-filter>
6     </activity>
7 </application>
```

BLUEBOX

Ambiguity during Transformation

- *When parsed in Android, attributes are identified according to an id rather than based on the representing attribute name string*:

```
<public type="attr" name="name" id="0x01010003" />
```

BLUEBOX

Ambiguity during Transformation

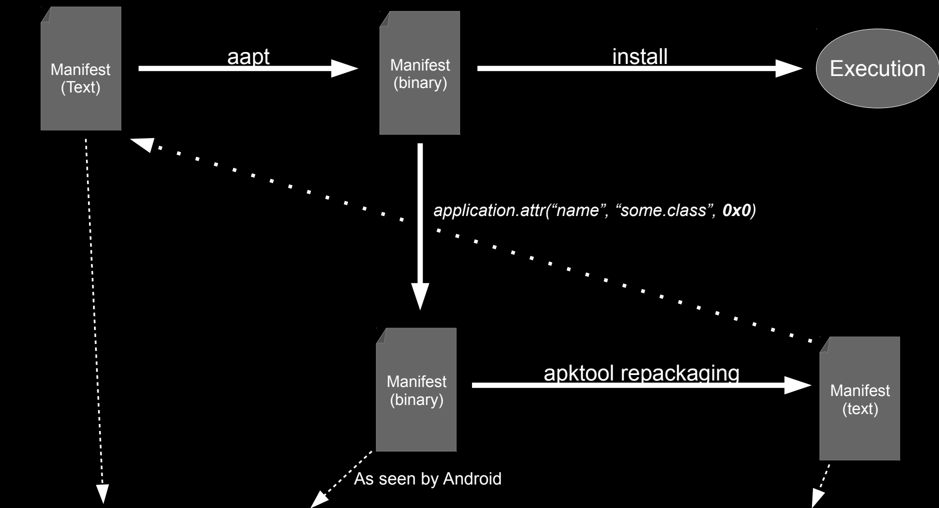
- *When parsed in Android, attributes are identified according to an id rather than based on the representing attribute name string*:

  ```
  <public type="attr" name="name" id="0x01010003" />
  ```

- Text -> Binary: Works just fine (aapt)
- Binary -> Text: Drops attribute id info (apktool)

$\rightarrow$ Inject a "name" attribute into <application> with an unknown id, so Android will not recognize it as a name attribute

BLUEBOX




```
<application>
   <activity android:name=
            "com.example.manifestexample.MainActivity">
      <intent−filter>
         <action android:name=
                  "android.intent.action.MAIN" / >
      </intent−filter >
   </activity>
</application>
```

```
<application android:name=”detect.class”>
   <activity android:name=
            "com.example.manifestexample.MainActivity">
      <intent−filter>
         <action android:name=
                  "android.intent.action.MAIN" / >
      </intent−filter >
   </activity>
</application>
```

Apply to existing application

1. Modify manifest by injecting a "name" attribute into the application tag with id 0 and value "detect.class"
2. **Android:** ignores the attribute, does not interpret as "android:name"
3. **Apktool** converts the binary xml into text; thus will include a proper "name" attribute when rebuilding the apk

Application output after repacking with apktool:

```
D/AndroidRuntime( 6387): Shutting down VM
W/dalvikvm( 6387): threadid=1: thread exiting with uncaught exception (group=0x41bd3930)
E/AndroidRuntime( 6387): FATAL EXCEPTION: main
E/AndroidRuntime( 6387): java.lang.RuntimeException: Unable to instantiate application some.class: jav
a.lang.ClassNotFoundException: Didn't find class "some.class" on path: /data/app/com.example.manifeste
xample-1.apk
E/AndroidRuntime( 6387):        at android.app.LoadedApk.makeApplication(LoadedApk.java:504)
E/AndroidRuntime( 6387):        at android.app.ActivityThread.handleBindApplication(ActivityThread.jav
a:4364)
```

BLUEBOX

Consequences

- Practical repack detection for apktool:
  1. Implement "detect.class"
  2. If it’s being executed, the app knows it has been repacked
- Android Application can read its own manifest, be creative ;)

BLUEBOX

Consequences

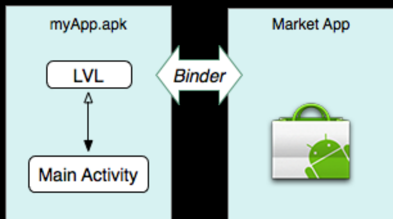
- Practical repack detection for apktool:
  1. Implement "detect.class"
  2. If it's being executed, the app knows it has been repacked
- Android Application can read its own manifest, be creative ;)

Android Binary-XML format is not properly representable using the Text-based form without additional metadata (recall: attribute id)

Runtime integrity checks

- Check app signature (signed by the developer)
- Try to do it on your own
- Use system services to check the signature

- Google Play Licensing Service
  `http://developer.android.com/google/play/licensing/overview.html`



BLUEBOX

Anti-Runtime Analysis

Anti-Runtime Analysis Layers

1. Detecting a debugger in Java
2. Detecting and preventing a debugger
   by interacting the Dalvik Virtual Machine directly

BLUEBOX

Example 1: Debugger Detection (Java)

```
1   static int detect_isDebuggerPresent(){
2           if(Debug.isDebuggerConnected())
3                   return 1;
4           else
5                   return 0;
6   }
```

BLUEBOX

Example 2: Debugger Detection (Java)

```java
static boolean detect_threadCpuTimeNanos(){
        long start = Debug.threadCpuTimeNanos();
        for(int i=0; i<1000000; ++i)
                continue;
        long stop  = Debug.threadCpuTimeNanos();
        if(stop - start < 10000000)
                return false;
        else
                return true;
}
```

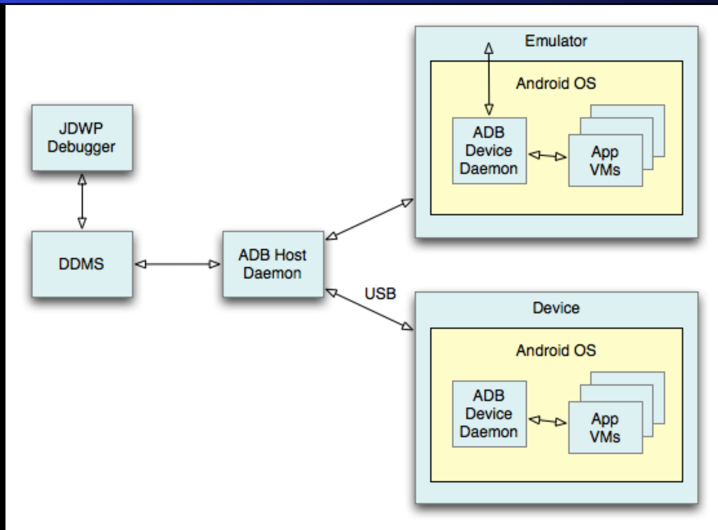
Example 3: Debugger Detection (Java)

```
1  static boolean detect_waitForDebugger(){
2          WaitForDebuggerThread thread = new WaitForDebuggerThread();
3          thread.start();
4          long start_ts = Calendar.getInstance().getTimeInMillis() / 1000;
5          long end_ts;
6          do {
7                  end_ts = Calendar.getInstance().getTimeInMillis() / 1000;
8                  long duration = end_ts - start_ts;
9                  if(duration > 1)
10                         return false;
11         } while(!WaitForDebuggerThread.done);
12         return true;
13 }
14
15 // public Class WaitForDebuggerThread
16 public void run(){
17         Debug.waitForDebugger();
18         done = true;
19 }
```

BLUEBOX

Debugger Detection (Native)

Android Application Debugging



http://developer.android.com/tools/debugging/index.html

BLUEBOX

Pro's and Con's compared to ptrace

- ✓ DVM implements debugging mechanisms
  - Breakpoints
  - Single-Stepping
  - Java object observation
  - Profiling
- × No OS in between
  - Additional Debug-Thread inside DVM
  - State tracking is done in the application context
  - Debugger communicates with the application directly instead of the OS

BLUEBOX

Pro's and Con's compared to ptrace

✓ DVM implements debugging mechanisms

- Breakpoints
- Single-Stepping
- Java object observation
- Profiling

× No OS in between

- Additional Debug-Thread inside DVM
- State tracking is done in the application context
- Debugger communicates with the application directly instead of the OS

→ What if the application denies following debugging protocols?

Relevant Data Structure

```
1   struct DvmGlobals {
2           /* ... */
3           bool debuggerConnected;
4           bool debuggerActive;
5           JdwpState* jdwpState;    // jdwp connection state
6           HashTable* dbgRegistry;  // object tracking
7           BreakpointSet*  breakpointSet;
8
9           // org.apache.harmony.dalvik.ddmc.DdmServer
10          Method* methDalvikDdmcServer_dispatch;
11          /* ... */
12  }
13  extern struct DvmGlobals gDvm;
```

BLUEBOX

Example: Debugger Detection (Native)

```
1 JNIEXPORT jboolean JNICALL Java_poc_c_detectdebuggerConnected(JNIEnv* env,
2                                                                jobject dontuse
3         if(gDvm.debuggerConnected || gDvm.debuggerActive)
4                 return JNI_TRUE;
5         return JNI_FALSE;
6 }
```

BLUEBOX

Example 1: Debugger Prevention

```
1  JNIEXPORT jboolean JNICALL Java_poc_c_crashOnInit(JNIEnv* env,
2                                                    jobject dontuse){
3          gDvm.methDalvikDdmcServer_dispatch = NULL;
4  }
```

- Crashes the debugging thread upon initialization
- Point to a valid location and have fun implementing your own endpoint

BLUEBOX

Example 2: Debugger Prevention

```
1 JNIEXPORT jboolean JNICALL Java_poc_c_crashOnBreakpoint(JNIEnv* env,
2                                                         jobject dontuse){
3         gDvm.breakpointSet = NULL;
4 }
```

- Crashes the debugging thread upon breakpoint usage

BLUEBOX

Example 3: Debugger Manipulation

```
1 JNIEXPORT jboolean JNICALL Java_poc_c_paralyseDebugger(JNIEnv* env,
2                                                        jobject dontuse){
3         dvmHashTableLock(gDvm.dbgRegistry);
4         dvmHashTableFree(gDvm.dbgRegistry);
5         gDvm.dbgRegistry = dvmHashTableCreate(1000, NULL);
6         dvmHashTableUnlock(gDvm.dbgRegistry);
7         return JNI_TRUE;
8 }
```

- Free all references to tracked objects

Anti-Debugging comparison

- Java based
  - ✓ trivial
  - ✓ stable
  - × Not many different methods (yet?)
- Native code based
  - ✓ Variety of methods nearly unlimited (be creative)
  - ✓ Enables crashing or manipulating the debugger
  - × Relatively easy to isolate code due to JNI interfacing

Conclusion

- ✓ Protect Android applications from being easily RE'd
- ✓ Pitfalls in Android application analysis

- Therefore we've presented some ideas including:
  - Callgraph obfuscation
  - Dynamic bytecode loading
  - Static repack protection using "Manifest Cheating"
  - Runtime integrity checks
  - Anti-Debugging using Java and native code

BLUEBOX

Find us...

please find us at

www.bluebox.com

{felix|patrick}@bluebox.com