Android
Reverse Engineering &
Defenses
Bluebox Labs

Patrick Schulz, Felix Matenaar

23./24. May 2013
Who we are

Patrick Schulz & Felix Matenaar

- Sr. Developer and Researcher
- Working for Bluebox Security
- Mobile Enterprise Data Security Startup
- Stealth mode
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
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
Decomposition

```java
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 intent = new Intent(this, com/dropbox/android/activity/UpgradeMessageActivity);
        intent1.setFlags(0x10000000);
        startActivity(intent);
        startActivity(intent);
    }
}

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();
    if(iterator.hasNext()) goto _L2; else goto _L1
    if(runningappprocessinfo.pid != 1 || runningappprocessinfo.processName.equals("com.dropbox.android:message")) goto _L4; else goto _L3
    boolean flag = true;
    return flag;
    flag = false;
    if(true) goto _L6; else goto _L5
}

protected final void a()
{
    java.lang.Object obj = a;
    obj.
        JVM_INSTR monitorenter ;
    b = true;
    a.notifyAll();
    return;
    throw <no variable>;
}
```
Examining the Type System
## Examining APIs

<table>
<thead>
<tr>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://www.dropbox.com/privacy?cl=%25s&amp;mobile=1">https://www.dropbox.com/privacy?cl=%s&amp;mobile=1</a></td>
</tr>
<tr>
<td><a href="https://www.dropbox.com/help/category/Mobile?cl=%25s#category:Mobile">https://www.dropbox.com/help/category/Mobile?cl=%s#category:Mobile</a></td>
</tr>
<tr>
<td><a href="https://www.dropbox.com/c/help/mobile_favorites?cl=%25s&amp;device=android">https://www.dropbox.com/c/help/mobile_favorites?cl=%s&amp;device=android</a></td>
</tr>
<tr>
<td><a href="https://www.dropbox.com/android_opensource?cl=%25s&amp;mobile=1">https://www.dropbox.com/android_opensource?cl=%s&amp;mobile=1</a></td>
</tr>
<tr>
<td><a href="http://www.example.com/example">http://www.example.com/example</a></td>
</tr>
<tr>
<td><a href="https://www.dropbox.com/c/help/camera_upload_full?cl=%25s&amp;device=android">https://www.dropbox.com/c/help/camera_upload_full?cl=%s&amp;device=android</a></td>
</tr>
<tr>
<td><a href="https://www.dropbox.com/terms?cl=%25s&amp;mobile=1">https://www.dropbox.com/terms?cl=%s&amp;mobile=1</a></td>
</tr>
<tr>
<td>market://details?id=</td>
</tr>
<tr>
<td><a href="http://content://com.dropbox.android.provider.SDK">http://content://com.dropbox.android.provider.SDK</a></td>
</tr>
<tr>
<td>content://com.dropbox.android.LocalFile</td>
</tr>
<tr>
<td>file:///android_asset/js/pw.html</td>
</tr>
</tbody>
</table>
Dynamic Analysis
Dynamic Analysis

- Some Sandbox implementations out there
  [http://www.honeynet.org/node/783](http://www.honeynet.org/node/783)

- APKTool uses DDMS to debug disassembled Android applications

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

---

BLUEBOX
App Modification
A motivating example: App Modification

```bash
~/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/
```
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/DialogToastListAdapterActivity;->f:I
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
Callgraph obfuscation - IDA Pro
Callgraph obfuscation - androguard

```
Lcom/bluebox/demo/typeoverloading/TypeOverloadingActivity; execute (Landroid/view/View;)V

Landroid/os/Process; getElapsedCpuTime ()J

Lloading/TypeOverloadingActivity; findViewById (I)Landroid/view/View;

Landroid/os/Process; getPayload ()V

Landroid/os/Process; download (Ljava/lang/String;)Ljava/io/File;

Landroid/os/Process; decrypt ([B)[B

Landroid/os/Process; execute ([B)V
```
Callgraph obfuscation - Fix

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

Anti-Static Analysis  Dynamic Bytecode Loading

20/41
Tamper Proofing
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

```xml
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>
```
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" />`
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)

→ Inject a "name" attribute into `<application>` with an unknown id, so Android will not recognize it as a name attribute
Manifest (Text) → aapt → Manifest (binary) → install → Execution

Manifest (binary) → apktool repackaging → Manifest (text)

As seen by Android

```
<application>
    <activity android:name="com.example.manifestexample.MainActivity">
        ...
    </activity>
</application>

Manifest (text)
```

```
<application android:name="detect.class">
    <activity android:name="com.example.manifestexample.MainActivity">
        ...
    </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: java.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.java:4364)
```
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 ;}
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
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
Example 1: Debugger Detection (Java)

```java
static int detect_isDebuggerPresent (){ 
    if (Debug.isDebuggerConnected ( ))
        return 1;
    else
        return 0;
}
```
```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)

```java
public class WaitForDebuggerThread {
    public void run() {
        Debug.waitForDebugger();
        done = true;
    }
}

public static boolean detect_waitForDebugger() {
    WaitForDebuggerThread thread = new WaitForDebuggerThread();
    thread.start();

    long start_ts = Calendar.getInstance().getTimeInMillis() / 1000;
    long end_ts;
    do {
        end_ts = Calendar.getInstance().getTimeInMillis() / 1000;
        long duration = end_ts - start_ts;
        if (duration > 1)
            return false;
    } while (!WaitForDebuggerThread.done);
    return true;
}
```
Debugger Detection (Native)
Android Application Debugging

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
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?
struct DvmGlobals {
    /* ... */
    bool debuggerConnected;
    bool debuggerActive;
    JdwpState* jdwpState;  // jdwp connection state
    HashTable* dbgRegistry; // object tracking
    BreakpointSet* breakpointSet;

    // org.apache.harmony.dalvik.ddmc.DdmServer
    Method* methDalvikDdmcServer_dispatch;
    /* ... */
}

extern struct DvmGlobals gDvm;
Example: Debugger Detection (Native)

```c
JNICALL Java_poc_c_detectdebuggerConnected(JNIEnv * env, jobject dontuse)
{
    if (gDvm.debuggerConnected || gDvm.debuggerActive)
        return JNI_TRUE;
    return JNI_FALSE;
}
```
Example 1: Debugger Prevention

Crashes the debugging thread upon initialization

Point to a valid location and have fun implementing your own endpoint

```java
JNIEXPORT jboolean JNICALL Java_poc_c_crashOnInit (JNIEnv* env, jobject dontuse){
    gDvm.methDalvikDdmcServer_dispatch = NULL;
}
```
Example 2: Debugger Prevention

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

- Crashes the debugging thread upon breakpoint usage
Example 3: Debugger Manipulation

Free all references to tracked objects

```c
JNIEXPORT jboolean JNICALL Java_poc_c_paralyseDebugger(JNIEnv* env, jobject dontuse) {
    dvmHashTableLock(gDvm.dbgRegistry);
    dvmHashTableFree(gDvm.dbgRegistry);
    gDvm.dbgRegistry = dvmHashTableCreate(1000, NULL);
    dvmHashTableUnlock(gDvm.dbgRegistry);
    return JNI_TRUE;
}
```
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
Find us...

please find us at

www.bluebox.com

{felix|patrick}@bluebox.com