Android Insider Attacks

David Goeth

University of Passau
Advanced seminar: Real Life Security
WS 2017/2018

January 19, 2018

Overview

Native Code

- Executable machine code
- Communication done via Java Native Interface (JNI) [Ora17]
- On Android: Native Development Kit (NDK) [Goo17a]
- Useful for:
  - Performance critical code
  - Using platform specific features
  - (Re)using native libraries

Java Native Interface

- Enables Java code to interoperate with applications and libraries written in other programming languages
- No security checks performed
- Java and Native code are running in the same address space
- Native Code is able to read and write arbitrary memory from the JVM

$\Rightarrow$ Dangerous seen from a security perspective.

Java Native Interface

Part 2

- Flaws in native libraries can enable attackers to read and write the JVM's memory [ST12, p. 3]
- JNI allows to retrieve and set the content of private fields. This enables an attacker to steal confidential information [ST12, p. 3]
- JNI allows to set the destination of an object pointer without type-checking. Thus type-confusion attacks are possible [ST12, p. 4]
- Bugs in the Linux kernel can enable an attacker to escalate privileges, like done in 'Dirty Cow' (CVE–2016–5195) [Oes16]. Over JNI such attacks can directly be initiated
- Changing the behavior of the Java application by replacing functions by manipulating or injecting byte code using code patching methods.

Java Native Interface

Part 3

Loading native library (in Java):

```
1  System.loadLibrary("libName");
```

Declaring a native method (in Java):

```
1 package com.example;
2 public class Native {
3   public native String fromNative();
4 }
```

Defining a native method (in C++):

```
1 extern "C" JNIEXPORT jstring JNICALL
2 Java_com_example_fromNative(JNIEnv *env,jobject /* this */) {
3     const char* msg = "Hello from C++";
4     return env->NewStringUTF(msg);
5 }
```

- JNIEnv defines the JNI interface methods.
- The JNIEnv variable is created by the JVM and a pointer to it is pushed on the callstack as the first parameter

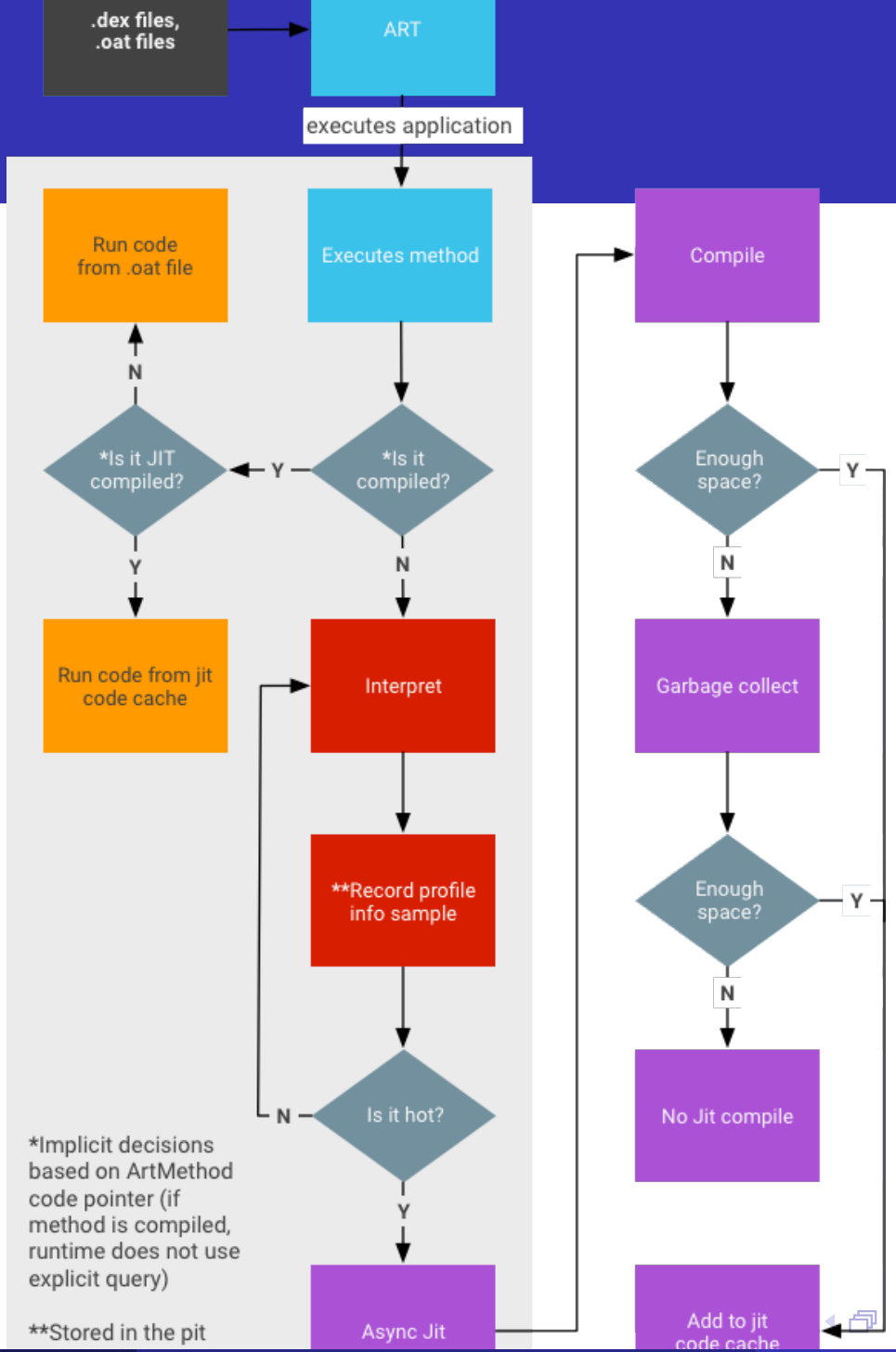
Android Runtime (ART)

- The managed runtime used by applications and some system services on Android ⇒ The "JVM" of Android
- Replaced Dalvik in Android 5.0 (Lollipop) [1]
- Introduced Ahead-of-time (AOT) compilation
- Android 7.0 (Nougat) reintroduced a JIT Compiler working along with the AOT compiler
  - Only 'hot' methods are compiled Ahead-of-time
  - If a method gets 'hot', it will be compiled by the AOT compiler
  - Native methods are not interpreted, 'hotness' count is ignored
    ⇒ In order to replace an existing java method, the method has to be set to native and its 'hotness' count has to be set to 0 (not 'hot')

[1] `https://developer.android.com/about/versions/android-5.0-changes.html`

ART JIT

Part 2



Function Hooking

- Intercepts a function and redirects execution flow to another memory address
- Established by using a jump or return instructions
- Save the original function before modifying it
  - Original function can still be called
  - Called trampoline hook
  - Technique coined by Micrsoft's Detour library [HB99]

Function Hooking (Art)

```
unsigned char redirect[] = {
        0x68, 0x74, 0x56, 0x34 , 0x12, // push   0x12345678 (
    addr that contains target address)
        0xc3                    //ret
};

void source() {}
void target() {}
void foo() {

  void* address = target;

  // set the target address inside redirect
  memcpy(redirect + 1, &address, 4);

  //Direct source to target
  memcpy(source, redirect, sizeof(redirect));
};

```

Listing 1: Redirection to another address

Back to JNI

- Java Methods and classes are mapped to C++ types
- Over JNIEnv, methods and classes can be found by their name (FindClass, GetMethodID)
- The definition of the structures are dependent from the concrete JVM implementation

```
1 struct _jmethodID;   /* opaque structure */
2 typedef struct _jmethodID* jmethodID;   /* method IDs */
```

Listing 2: jmethodID definition in jni.h

- In Android 8.0.0 (rev. 36) the definition is in art/runtime/art_method.h [Goo18]
  ⇒ jmethodID is actually a pointer to an ArtMethod object!

Function Hooking (Art)

Part 4

```
1
2  class ArtMethod FINAL {
3   protected:
4    GcRoot<mirror::Class> declaring_class_; //offset 0
5    std::atomic¡std::uint32_t¿ access_flags_; //offset 4
6    uint32_t dex_code_item_offset_; //offset 8
7    uint32_t dex_method_index_; //offset 12
8    uint16_t method_index_; //offset 16
9    uint16_t hotness_count_; //offset 18
10   struct PtrSizedFields {
11     ArtMethod** dex_cache_resolved_methods_; //offset 20
12     void* data_; //offset 24
13     void* entry_point_from_quick_compiled_code_; //offset 28
14   } ptr_sized_fields_;
15 };
```

Listing 3: ArtMethod and x86 offsets [Goo18]

Function Hooking (Art)

Part 5

- entry_point_from_quick_compiled_code_ is used as the entry point for all methods
- data_ is used for secveral purposes, but for native functions it points to the native code.
- access_flags_ contains a specific flag for native methods
- hotness_count_ is not used for native methods
- hotness_count_ has to be set to 0 for hooked methods not being native before the method is called

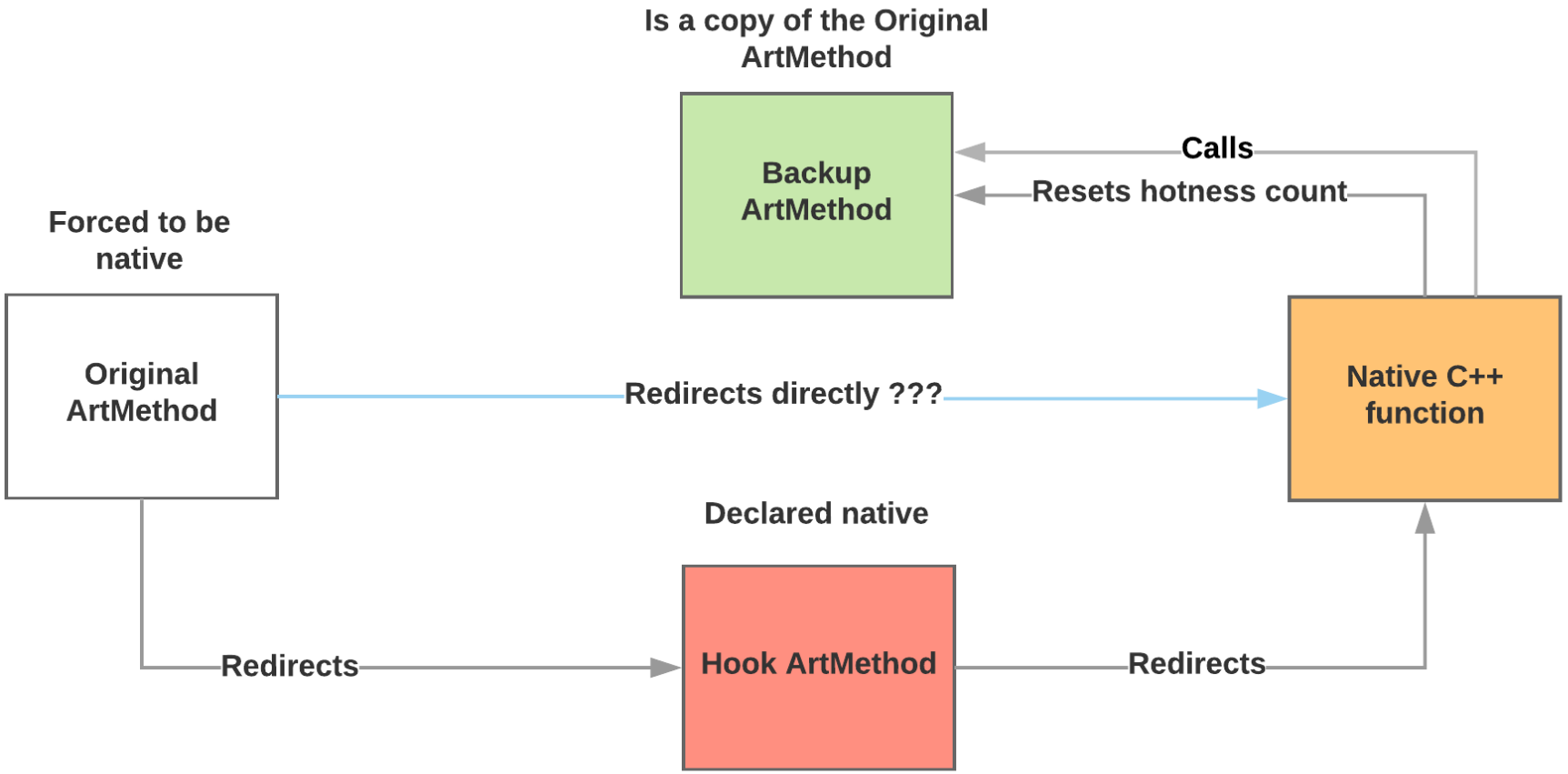


Figure: Hooking an Art method

Attack Scenario

Benign App:

- Loads a native library
- A simple text messenger
- User can write text messages to a remote server
- Communication is encrypted using asymmetric cryptography (TLSv1.2)

Evil Library:

- Wants to know what is send and received
- Downloads helping dex file from evil remote server that contains method declarations for the function hooks
- Hooks send and receive methods of the benign app
- Changes the content of send and received messages

References I


Google, *Android ndk intro*, https://developer.android.com/ndk/guides/index.html, 2017, Last accessed on 11/10/2017.


_____, *Implementing art just-in-time (jit) compiler*, https://source.android.com/devices/tech/dalvik/jit-compiler, 2017, Last accessed on 01/18/2018.


_____, *Artmethod from goolge source*, https://android.googlesource.com/platform/art/+/android-8.0.0_r36/runtime/art_method.h, 2018, Last accessed on 01/18/2018.


Galen Hunt and Doug Brubacher, *Detours: Binary interception of win32 functions*, Third USENIX Windows NT Symposium, USENIX, July 1999, p. 8.

References II


Phil Oester, *Cve-2016-5195*, `https://access.redhat.com/security/cve/CVE-2016-5195`, 2016, Last accessed on 01/17/2018.


Oracle, *Jni specification chapter 1*, `https://docs.oracle.com/javase/8/docs/technotes/guides/jni/spec/intro.html`, 2017, Last accessed on 11/13/2017.


Mengtao Sun and Gang Tan, *Jvm-portable sandboxing of java's native libraries*, pp. 842–858, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.