Google Play Protect

Unpacking the Packed Unpacker

Reversing an Android Anti-Analysis Native Library

Maddie Stone
@maddiestone
BlackHat USA 2018

Who am I? - Maddie Stone

- Reverse Engineer on Google's Android Security Team
- 5+ years hardware & firmware reversing of embedded devices
- Creator of IDAPython Embedded Toolkit
- BS in Computer Science, Russian, & Applied Math
- MS in Computer Science

@maddiestone

Malware Analysts vs Malware Authors

striving for the asymmetric advantage

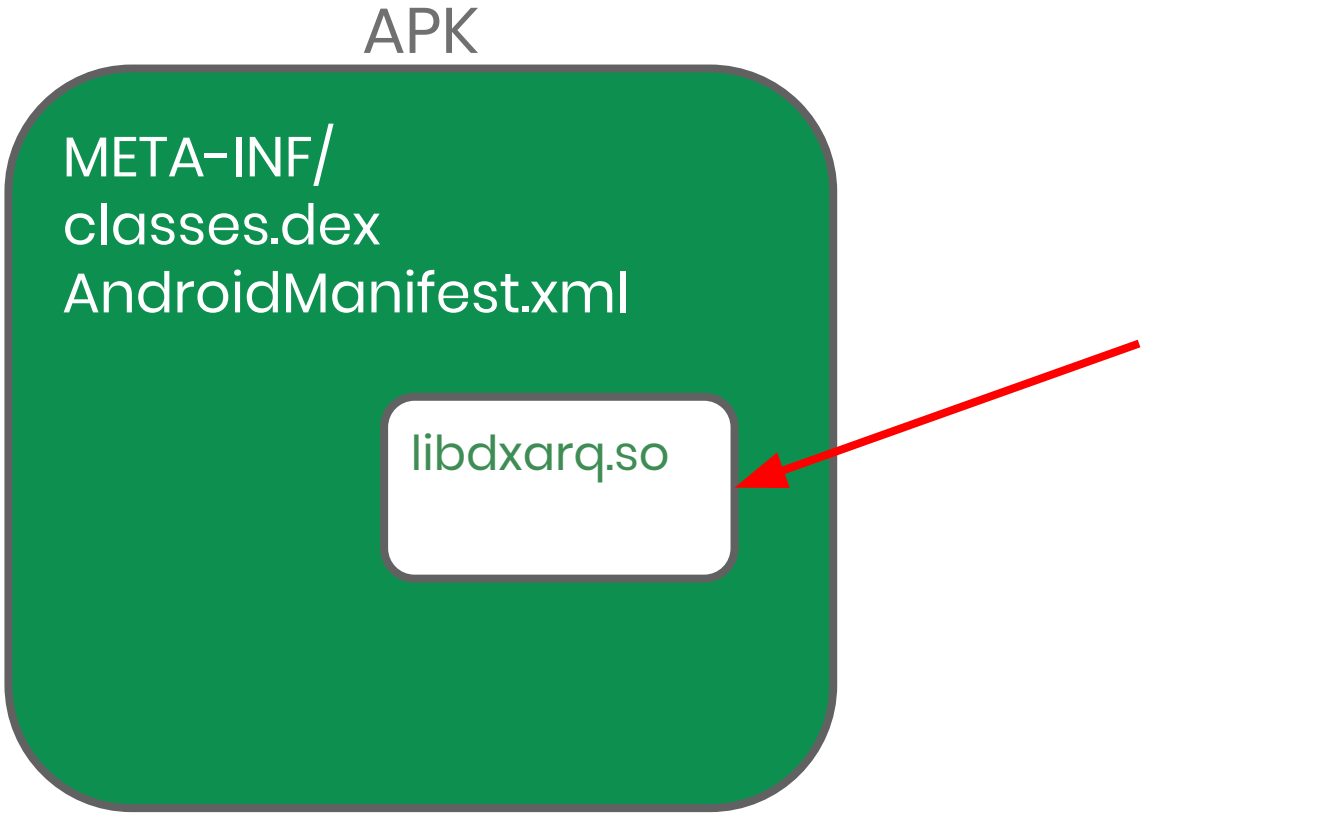
Anti-analysis: techniques to frustrate analysis and make reverse engineering malware more difficult

Analyst Challenge

Objective: Determine if an app is malware. Quickly.

- Have an app that looks suspicious, but need evidence to determine if it's malware
  - App won't run in dynamic analysis
  - Most code is native
  - Many similar apps

Introduction - Target of Analysis



Target of Analysis: Android App Native Library

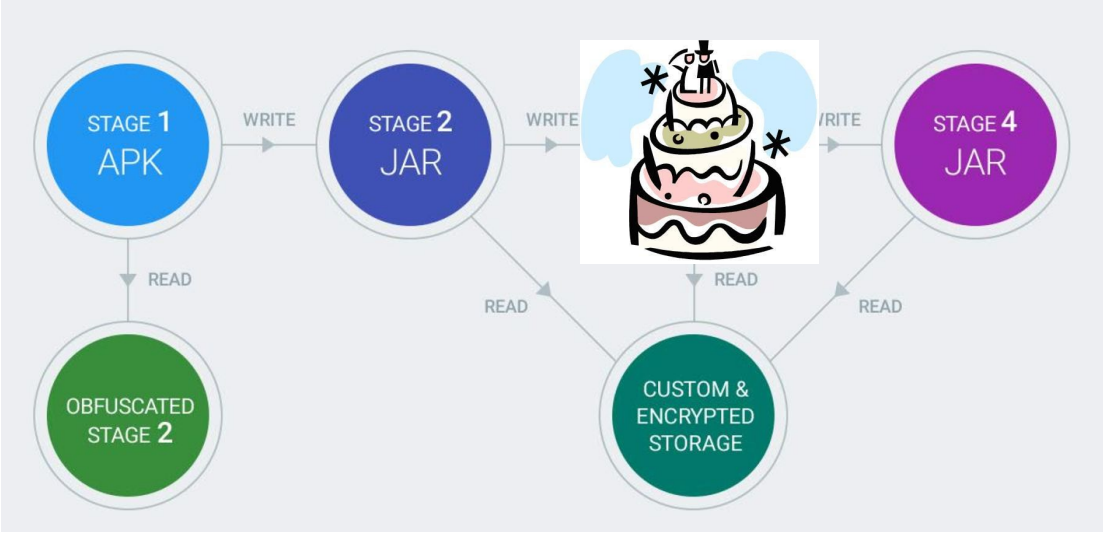
(*.so in ELF format)

Introducing WeddingCake!

...because it has lots of layers

Purpose of WeddingCake

- 5000+ distinct APK samples containing WeddingCake
- Used by newer variants of Chamois family
- Protects functionality that authors want to hide by "wrapping" it in anti-analysis protections



WeddingCake Anti-Analysis Techniques

JNI Manipulations

Anti-Reverse Engineering

In-Place Decryption

Run-Time Environment Checks

- Check System Properties
- Verifying CPU architecture
- Check for Monkey
- Check for Xposed Framework

Characteristics of WeddingCake

- Android native library included in APKs as .so/ELF
- Different name in each sample
  `lib[3-8 random lowercase characters].so`

- Java classes that interface with library have random names -> different for each sample
  `ses.fdkxxcr.udayjfrgxp.ojoyqmosj.xien.xmdowmbkdgfgk`

- Two strings under the .comment section in the ELF:
  `Android clang version 3.8.275480 (based on LLVM 3.8.275480)`
  `GCC: (GNU) 4.9.x 20150123 (prerelease)`

Characteristics of WeddingCake

- Two Java-declared native methods with the following signatures
  - **The names of the methods change in each sample**

```
public native int vxeg(Object[] p0);
```

Performs run-time environment checks and the ELF's main functionality

```
public static native String quaqrd(int p0);
```

Returns string at index p0 in a hard-coded array

- Samples often have a 3rd method declared:

```
public native Object ixkjwu(Object[] p0);
```

CPU Variants of WeddingCake

- 32-bit “generic” ARM is most common (`armeabi`)
- Also seen other versions of the library:
  - 32-bit ARMv7 (`armeabi-v7a`)
  - ARM64 (`arm64-v8a`)
  - x86 (`x86`)

`92e80872cfd49f33c63993d52290afd2e87cbef5db4adff1bfa97297340f23e0`
https://bit.ly/2Ll8eT0

Anti-Analysis Lib File Paths	Anti-Analysis Library “Type”
`lib/armeabi/librxovdx.so`	32-bit “generic” ARM
`lib/armeabi-v7a/librxovdx.so`	32-bit ARMv7
`lib/x86/libaojjp.so`	x86

Analyzing WeddingCake

Sample:

`e8e1bc048ef123a9757a9b27d1bf53c092352a26bdbf9fbdc10109415b5cadac`

`https://bit.ly/2Nkc4ZS`

Intro to Java Native Interface (JNI)

- JNI allows developers to declare Java native methods that run in other languages (C/C++) in the application

- Native methods are declared in Java

```
public static native String quaqrd(int p0);

public native Object ixkjwu(Object[] p0);

public native int vxeg(Object[] p0);
```

- The declared Java native method is implemented in C/C++

"Getting Started with the NDK", Android, https://developer.android.com/ndk/guides/

"JNI Tips", Android, https://developer.android.com/training/articles/perf-jni

Intro to Java Native Interface (JNI)

`System.loadLibrary("calc")`

or

`System.load("lib/armeabi/libcalc.so")`

When `load()` or `loadLibrary()` is called in Java, the ELF is "loaded" and `JNI_OnLoad()` is run in the ELF

Intro to Java Native Interface (JNI)

- "Registering" native methods: pair the Java method declaration to the correct subroutine in the native library
    - "Discovery": the function names and function signatures matching in both Java and the `.so`

      `Java_<mangled class name>_<mangled method_name>`
    - `RegisterNatives` JNI function
        - Requires string of the method name and the string of the method signature

"Resolving Native Method Names", Oracle, https://docs.oracle.com/javase/6/docs/technotes/guides/jni/spec/design.html#wp615

"Registering Native Methods in JNI", Stack Overflow, https://stackoverflow.com/questions/1010645/what-does-the-registernatives-method-do

Intro to Java Native Interface (JNI)

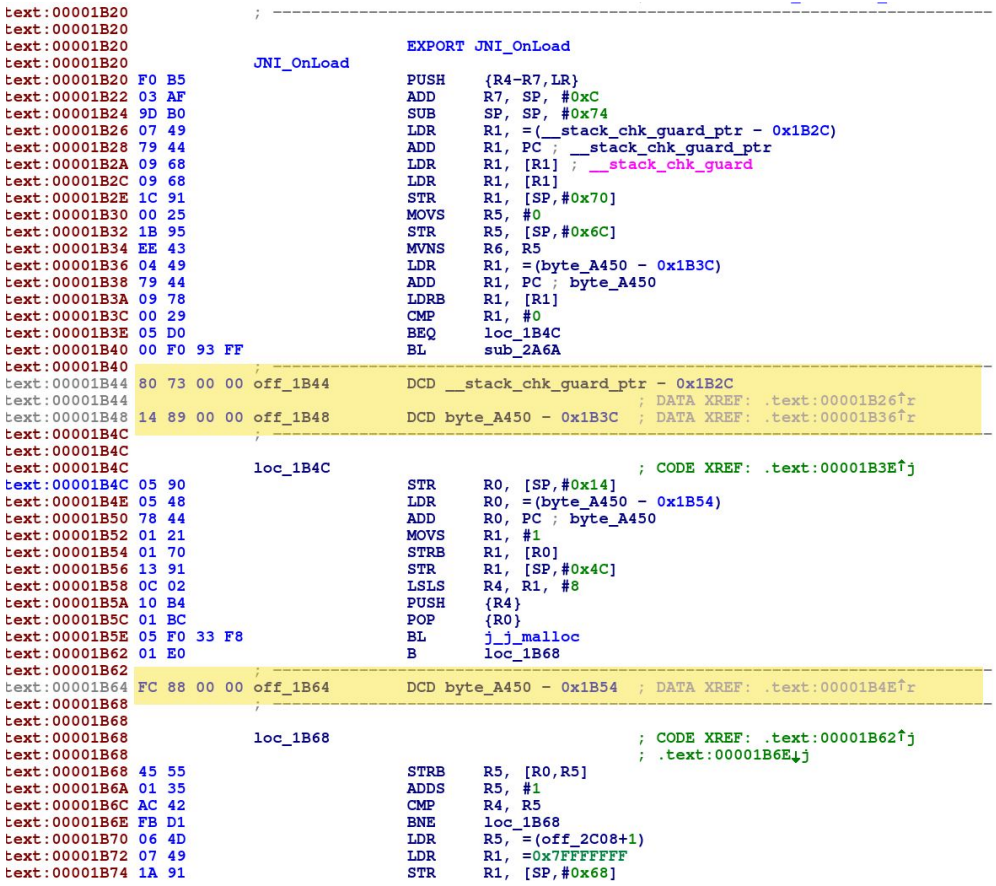
```
jint RegisterNatives(JNIEnv *env, jclass clazz,
const JNINativeMethod *methods, jint nMethods);

typedef struct {
    char *name;
    char *signature;
    void *fnPtr;
} JNINativeMethod;
```

- Signatures

```
public static native String quaqrd(int p0); ⟶
                                "(I)Ljava/lang/String;"
```

First Look

- None of the native method names exist in native lib (as funcs or strings)
- `JNI_OnLoad` is exported, but not defined in IDA
  - The bytes at `+0x24`, `+0x28`, and `+0x44` are defined as data
- No strings
  - Including method names and signatures



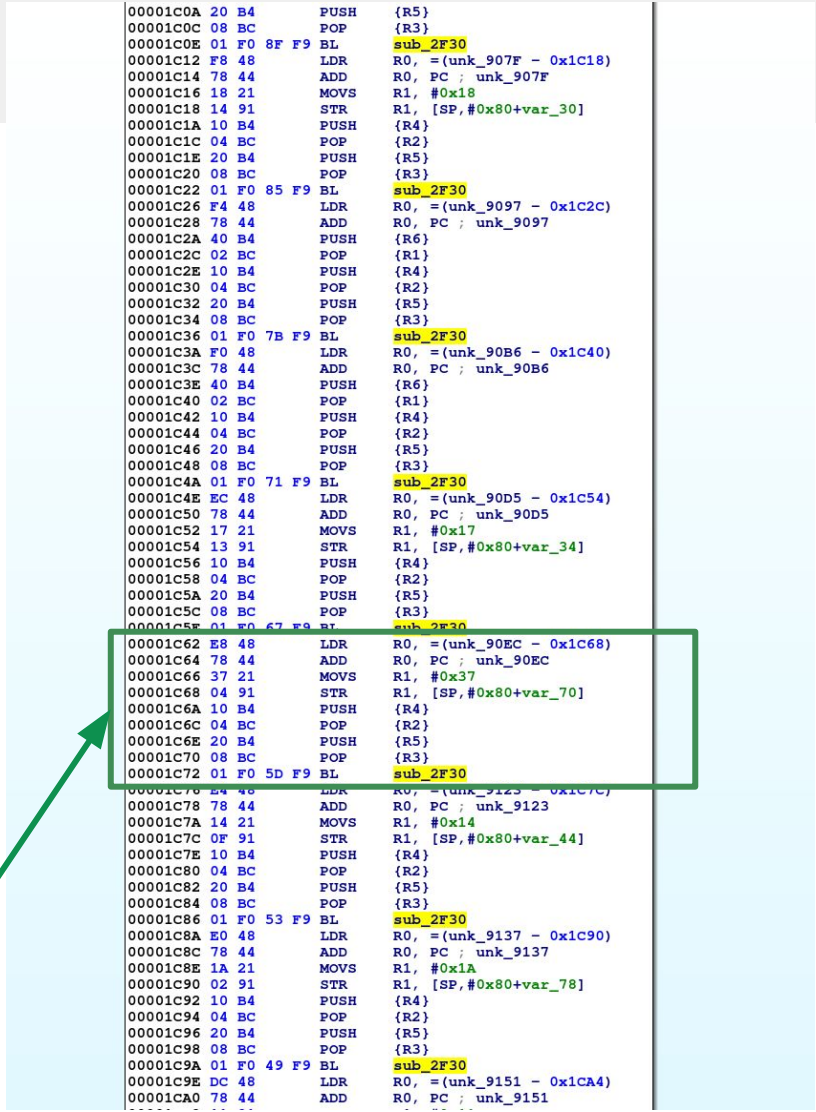
```
text:00001B20                 ; ---------------------------------------------------------------------------
text:00001B20
text:00001B20                                 EXPORT JNI_OnLoad
text:00001B20                 JNI_OnLoad
text:00001B20 F0 B5                           PUSH    {R4-R7,LR}
text:00001B22 03 AF                           ADD     R7, SP, #0xC
text:00001B24 9D B0                           SUB     SP, SP, #0x74
text:00001B26 07 49                           LDR     R1, =(__stack_chk_guard_ptr - 0x1B2C)
text:00001B28 79 44                           ADD     R1, PC ; __stack_chk_guard_ptr
text:00001B2A 09 68                           LDR     R1, [R1] ; __stack_chk_guard
text:00001B2C 09 68                           LDR     R1, [R1]
text:00001B2E 1C 91                           STR     R1, [SP,#0x70]
text:00001B30 00 25                           MOVS    R5, #0
text:00001B32 1B 95                           STR     R5, [SP,#0x6C]
text:00001B34 EE 43                           MVNS    R6, R5
text:00001B36 04 49                           LDR     R1, =(byte_A450 - 0x1B3C)
text:00001B38 79 44                           ADD     R1, PC ; byte_A450
text:00001B3A 09 78                           LDRB    R1, [R1]
text:00001B3C 00 29                           CMP     R1, #0
text:00001B3E 05 D0                           BEQ     loc_1B4C
text:00001B40 00 F0 93 FF                     BL      sub_2A6A
text:00001B40                 ; ---------------------------------------------------------------------------
text:00001B44 80 73 00 00     off_1B44        DCD __stack_chk_guard_ptr - 0x1B2C
text:00001B44                                                         ; DATA XREF: .text:00001B26↑r
text:00001B48 14 89 00 00     off_1B48        DCD byte_A450 - 0x1B3C  ; DATA XREF: .text:00001B36↑r
text:00001B4C                 ; ---------------------------------------------------------------------------
text:00001B4C
text:00001B4C                 loc_1B4C                                ; CODE XREF: .text:00001B3E↑j
text:00001B4C 05 90                           STR     R0, [SP,#0x14]
text:00001B4E 05 48                           LDR     R0, =(byte_A450 - 0x1B54)
text:00001B50 78 44                           ADD     R0, PC ; byte_A450
text:00001B52 01 21                           MOVS    R1, #1
text:00001B54 01 70                           STRB    R1, [R0]
text:00001B56 13 91                           STR     R1, [SP,#0x4C]
text:00001B58 0C 02                           LSLS    R4, R1, #8
text:00001B5A 10 B4                           PUSH    {R4}
text:00001B5C 01 BC                           POP     {R0}
text:00001B5E 05 F0 33 F8                     BL      j_j_malloc
text:00001B62 01 E0                           B       loc_1B68
text:00001B62                 ; ---------------------------------------------------------------------------
text:00001B64 FC 88 00 00     off_1B64        DCD byte_A450 - 0x1B54  ; DATA XREF: .text:00001B4E↑r
text:00001B68                 ; ---------------------------------------------------------------------------
text:00001B68
text:00001B68                 loc_1B68                                ; CODE XREF: .text:00001B62↑j
text:00001B68                                                         ; .text:00001B6E↓j
text:00001B68 45 55                           STRB    R5, [R0,R5]
text:00001B6A 01 35                           ADDS    R5, #1
text:00001B6C AC 42                           CMP     R4, R5
text:00001B6E FB D1                           BNE     loc_1B68
text:00001B70 06 4D                           LDR     R5, =(off_2C08+1)
text:00001B72 07 49                           LDR     R1, =0x7FFFFFFF
text:00001B74 1A 91                           STR     R1, [SP,#0x68]
```

Beginning Analysis

- Start with `JNI_OnLoad`
- Repetitive calls to same function over different blocks of memory → **Encryption**

`sub_2F30`: Decryption Subroutine

```
00001C62 E8 48        LDR     R0, =(unk_90EC - 0x1C68)
00001C64 78 44        ADD     R0, PC ; unk_90EC
00001C66 37 21        MOVS    R1, #0x37
00001C68 04 91        STR     R1, [SP,#0x80+var_70]
00001C6A 10 B4        PUSH    {R4}
00001C6C 04 BC        POP     {R2}
00001C6E 20 B4        PUSH    {R5}
00001C70 08 BC        POP     {R3}
00001C72 01 F0 5D F9  BL      sub_2F30
```



In-Place Decryption

```
sub_2F30(Byte[] encrypted_array, int length, Word[]
word_seed_array, Byte[] byte_seed_array)
```

- `encrypted_array`: Pointer to the encrypted byte array (bytes to be decrypted)
- `length`: Length of the encrypted byte array
- `word_seed_array`: Word (each value in array is 4 bytes) seed array
- `byte_seed_array`: Byte (each value in array is 1 byte) seed array

Generating the Seed Arrays

```
byte_seed_array = malloc(0x100u);
index = 0;
 do
 {  byte_seed_array[index] = index;
    ++index;  }
 while ( 256 != index );
 v4 = 0x2C09;
 curr_count = 256;
 copy_byte_seed_array = byte_seed_array
 do
 {
     v6 = 0x41C64E6D * v4 + 0x3039;
     v7 = v6;
     v8 = copy_byte_seed_array[v6];
     v9 = 0x41C64E6D * (v6 & 0x7FFFFFFF) + 0x3039;
     copy_byte_seed_array[v7] = copy_byte_seed_array[v9];
     copy_byte_seed_array[v9] = v8;
     --curr_count;
     v4 = v9 & 0x7FFFFFFF;
}
while ( curr_count );
word_seed_array = malloc(0x400u);
index = 0;
do
{
  word_seed_array[byte_seed_array[index]] =
     index;
  ++index;
}
while ( 256 != index );
```

Generating the Seed Array: Anti-Reversing

- Output of the Seed Array Generation Algorithms:
  - Byte Seed Array - byte array from 0 to 0xFF
  - Word Seed Array - word (4 bytes) array from 0 to 0xFF

- Anti-Reversing Technique
  - Complex algorithm instead of simple algorithm

- Bypass:
  - Run dynamically and capture arrays

Decryption Algorithm

- The overall framework of the in-place decryption process is:
  1) Decryption function is called on an array of encrypted bytes
  2) Decryption is performed
  3) The encrypted bytes are overwritten by the decryption bytes

- Not identified as any known encryption/decryption algorithm

Decrypting the Library

- Need to decrypt the native library quickly for further analysis
  - Don't need to understand the decryption → just need to build a solution to decrypt it
  - "Translate" the decryption function to something that can run over the ELF
- Want any solution to be applicable to the multitude of samples
  - Different memory address, registers

IDAPython Decryption Script:

http://www.github.com/maddiestone/IDAPythonEmbeddedToolkit/Android/WeddingCake_decrypt.py

“Translating” the Decryption to IDAPython

ASM

```
0x2F4E 04 20     MOVS   R0, #4
0x2F50 C4 43     MVNS   R4, R0
0x2F52 00 22     MOVS   R2, #0
0x2F54 04 B4     PUSH   {R2}
0x2F56 20 BC     POP    {R5}
```

PYTHON

```
reg_0 = 4
reg_4 = ~(0x00000004)
reg_2 = 0
reg_5 = 0
```

Decryption Demo

Developing Decryption Solutions

- Speed
  - “Translating” to a format that can run over ELF instead of reversing for understanding
    - Python, ARM (or other CPU) emulators, etc.

- Flexibility
  - Doesn’t have to be modified to run over different samples
    - RegEx instead of hard-coded addresses/registers
    - Dynamically read bytes to be decrypted

Decrypted Contents

Each of the encrypted arrays decrypts to a string

```
00009480  01 F5 F0 81 88 94 F1 C6  29 18 2F DD 0C 34 AE 32  ........)./..4.2
00009490  EA 8E 53 58 0C 52 EE BE  2F 05 F5 0F C2 FC 18 BA  ..SX.R../.......
000094A0  B3 6E 36 39 C7 D2 FD D5  FE 73 4B 3A A3 06 FE D3  .n69.....sK:....
000094B0  F5 88 46 0A DC 14 28 D8  CB 5D 59 44 EB 2E FD A2  ..F...(..]YD....
000094C0  F0 8C 56 61 E1 F2 36 E2  1A 91 2C 65 ED 31 3E EE  ..Va..6...,e.1>.
000094D0  17 A6 34 48 F5 47 42 00  20 75 0D CD C2 56 98 9E  ..4H.GB. u...V..
000094E0  54 7E FB 08 59 47 3E E4  CC 0C FB 90 DE 5C FE 9B  T~..YG>......\..
000094F0  5F AA AF 63 67 D7 EF E5  C6 80 99 9B 94 1B 6F 24  _..cg.........o$
00009500  AB 22 DF 38 3A 0F 3D 84  A8 5E 94 7E D2 D0 6B 8F  ."..8:.=..^.~..k.
00009510  4E C8 D5 75 A4 89 D8 DF  3B 78 98 DD E4 76 A8 A2  N..u....;x...v..
00009520  C6 DA 89 AE 9F EF DF 8D  7F 38 15 5A 5A FA 22 05  .........8.ZZ."..
00009530  8B F9 88 E0 8A 00 4C E9  0B 9B 7D 91 8F BE 05 A2  ......L...}.....
00009540  DE 71 DE 3F 8E 25 67 25  CC DA 81 95 2B 44 33 0F  .q.?.%g%....+D3.
00009550  0D 52 3B 2E AA B6 E8 3C  AE 33 FB 4D EF 14 6E 2A  .R;....<.3.M..n*
00009560  11 D1 65 B2 E8 D6 44 B0  5F A2 49 48 EC 0E E3 29  ..e...D._.IH...)
00009570  1C 32 1C A2 E3 C7 2F F7  05 2A C3 EF 77 0A A9 37  .2..../..*..w..7
00009580  E9 EC 8A 01 7D 61 F7 03  8B 0E BB 4F B2 E3 92 07  ....}a.....O....
```

```
00009480  01 F5 F0 81 88 94 F1 C6  29 18 2F DD 0C 28 5B 42  ........)./..([B
00009490  29 5B 42 00 0C 52 EE BE  2F 05 28 4C 6A 61 76 61  )[B..R../.(Ljava
000094A0  2F 6C 61 6E 67 2F 53 74  72 69 6E 67 3B 5B 4C 6A  /lang/String;[Lj
000094B0  61 76 61 2F 6C 61 6E 67  2F 43 6C 61 73 73 3B 29  ava/lang/Class;)
000094C0  4C 6A 61 76 61 2F 6C 61  6E 67 2F 72 65 66 6C 65  Ljava/lang/refle
000094D0  63 74 2F 4D 65 74 68 6F  64 3B 00 CD C2 56 98 9E  ct/Method;...V..
000094E0  54 7E FB 08 6A 61 76 61  2F 6C 61 6E 67 2F 49 6E  T~..java/lang/In
000094F0  74 65 67 65 72 00 EF E5  C6 80 99 9B 94 1B 6F 24  teger.........o$
00009500  AB 22 DF 38 3A 0F 28 29  4C 61 6E 64 72 6F 69 64  ."..8:.()Landroid
00009510  2F 63 6F 6E 74 65 6E 74  2F 43 6F 6E 74 65 6E 74  /content/Content
00009520  52 65 73 6F 6C 76 65 72  3B 00 15 5A 5A FA 22 05  Resolver;..ZZ."..
00009530  4C 6A 61 76 61 2F 6C 61  6E 67 2F 53 74 72 69 6E  Ljava/lang/Strin
00009540  67 3B 00 3F 8E 25 67 25  CC DA 81 95 2B 44 33 0F  g;.?.%g%....+D3.
00009550  0D 28 29 5B 42 00 FB 60  63 7B 93 A1 9B C0 75 2A  .()[B..`c{....u*
00009560  11 D1 65 B2 E8 D6 44 B0  5F A2 49 48 EC 0E 41 45  ..e...D._.IH..AE
00009570  53 00 21 A2 E3 C7 2F F7  05 28 4C 6A 61 76 61 2F  S.!.../..(Ljava/
00009580  6C 61 6E 67 2F 53 74 72  69 6E 67 3B 29 4C 6A 61  lang/String;)Lja
00009590  76 61 2F 73 65 63 75 72  69 74 79 2F 4D 65 73 73  va/security/Mess
```

Decrypted Contents

```
off_9048        DCD aVxeg                   ; DATA XREF: .text:00002AAA↑
                                            ; .text:00002B0A↑o ...
                                            ; "vxeg"
                DCD aLjavaLangObj_0         ; "([Ljava/lang/Object;)I"
                DCD vxeg_sub_30D4+1
```

Decrypted Contents

	Native Function Name	Native Subroutine Address	Signature	Human-Readable Signature
1	vxeg	0x30D4	([Ljava/lang/Object;)I	public native int vxeg(Object[] p0);
2	quaqrd	0x4814	(I)Ljava/lang/String;	public static native String quaqrd(int p0);
3	ixkjwu	----	([Ljava/lang/Object;)Ljava/lang/Object;	public native Object ixkjwu(Object[] p0);

The method numbers in the left most column are used to identify the identical method in other samples where the method name is different, but the signature is the same

Run-Time Environment Checks

Run-Time Environment Checks

Goal: Detect if application is being dynamically analyzed, debugged, or emulated

The developers would rather limit the number of potential targets than risk being detected

Run-Time Environment Checks

- Function #1 (vxeg) performs the run-time environment checks
- 45+ run-time checks:
  - Checking system properties
  - Verifying CPU architecture by reading the `/system/lib/libc.so` ELF header
  - Looking for Monkey by iterating through all PIDs in `/proc`
  - Ensuring the Xposed Framework is not mapped to the application process memory
- If any one of these conditions is detected, the Linux exit(0) function is called, which terminates the Android application

System Property Checks

Goal: Check if system properties show that the “hardware” is an emulator or being debugged

- 37 system properties are checked for specific values
  - Mostly debugging & emulation checks
  - All at https://bit.ly/2KD5k7S
- 5 system properties are checked for existence

If any of these System Properties exist, the application exits
`init.svc.vbox86-setup`
`qemu.sf.fake_camera`
`init.svc.goldfish-logcat`
`init.svc.goldfish-setup`
`init.svc.qemud`

Verifying CPU Architecture

Goal: Ensure the application is running on ARM

- Read 0x14 bytes from `/system/lib/libc.so`
  - Reading the ELF header

- Verify one of the following conditions is true:

`e_ident[EI_CLASS] == 0x01` (32-bit) AND `e_machine == 0x0028` (ARM)

`e_ident[EI_CLASS] == 0x02` (64-bit) AND `e_machine == 0x00B7` (AArch64)

Identifying if Monkey is Running

Goal: Determine if application is run in emulator with “fake” user

- “The Monkey is a program that runs on your emulator or device and generates pseudo-random streams of user events such as clicks, touches, or gestures, as well as a number of system-level events.”
- Iterates through every PID directory under `/proc/` to determine if `com.android.commands.monkey` is running
  - Note that this no longer works on Android N+

Identifying if Monkey is Running

1. Verify `d_type` from the dirent `struct == DT_DIR`
2. Verify `d_name` from the dirent struct is an integer
3. Construct path strings: `/proc/[pid]/comm` and `/proc/[pid]/cmdline` where `[pid]` is the directory entry name that has been verified to be an integer
4. Attempts to read `0x7F` bytes from both `comm` and `cmdline` constructed path strings
5. Stores the data from whichever attempt (`comm` or `cmdline`) read more data
6. Checks if the read data equals `com.android.commands.monkey`, meaning that package is running

Current Process not Hooked with Xposed Framework

Goal: Confirm the application is not being analyzed and hooked with the Xposed Framework

- The Xposed Framework allows hooking and modifying of the system code running on an Android device
- Checks if `LIBXPOSED_ART.SO` or `XPOSEDBRIDGE.JAR` exist in `/proc/self/maps`
- Tries to find either of the following two classes using the JNI `FindClass()` method
  - XC_MethodHook: `de/robv/android/xposed/XC_MethodHook`
  - XposedBridge: `de/robv/android/xposed/XposedBridge`

Summary of Run-Time Environment Checks

JNI Manipulations

Anti-Reverse Engineering

In-Place Decryption

Run-Time Environment Checks

- Check System Properties
- Verifying CPU architecture
- Check for Xposed Framework
- Check for Monkey

Conclusion

Conclusion

Malware authors are willing to miss-out on potential targets if that means not being detected

- Layered Anti-Analysis Techniques:
  - Techniques that deter human analysis (anti-RE, decryption)
  - Techniques that prevent dynamic analysis (decryption)
  - Techniques that detect dynamic analysis, debugging, & emulation

Key Takeaways

- **Current Android anti-analysis techniques:** Android malware authors are becoming more sophisticated and implementing new techniques to hinder analysts and reverse engineers
- **Avoid anti-analysis techniques:** How to identify Android anti-analysis and anti-reversing traps in disassembly & how to avoid them, instead spending time and analytical resources on the malicious payload
- **Writing decryption solutions:** How to write a script to decrypt an ELF library with a focus on speed and flexibility (referring to memory addresses and registers) to reverse engineer more malicious payloads more quickly

THANK YOU

@maddiestone
github.com/maddiestone/IDAPythonEmbeddedToolkit


Google Play
Protect

Intro to Java Native Interface (JNI)

- In Java, call one of two methods to “load” the native library where the native method is implemented:
  - `System.loadLibrary()`
    - If the .so lives in the expected path `/lib/<cpu>/lib<name>.so → System.loadLibrary(<name>)`
    - Will automatically detect CPU and select appropriate library
  - `System.load()`
    - Requires the full path to the library
    - Developer manually selects which CPU version of the lib to load
- When `load` or `loadLibrary` is called in Java bytecode, `JNI_OnLoad` is run in the native library