Android - Bytecode Obfuscation

bringing x86 fuckups to dalvik

Patrick Schulz

thuxnder@dexlabs.org

06.07.2012

Overview

1. Introduction
2. Reverse Engineering Tools
3. Obfuscation techniques
4. Conclusion

Introduction

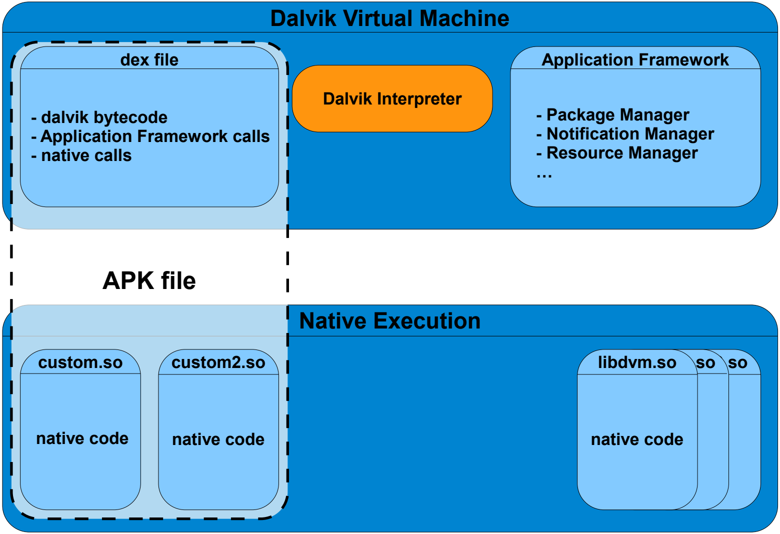
Android

- Operating System for mobile devices
- attractive research field
- Applications (dalvik/native/resources)

Reverse engineering

We want to analyse Android applications.

Application Runtime



Dalvik bytecode

- instructions of various size
- words-aligned (16-bit code unit)
- move, return, const, goto, if, invoke, binop, unop
- new-instance, fill-array, switch

Reverse Engineering Tools

Reverse Engineering Tools

- Disassembler
  - **dexdump:** c/cpp, Android SDK, meta information, stdout
  - **baksmali:** java, assembler, jasmin syntax, file output
  - **Dedexer:** java, jasmin syntax, file output
  - **Androguard:** python/cpp, cli
  - **IDA Pro:** closed source, gui, plugins
- Decompiler
  - **jad:** java decompiler, uses dex2jar
  - **jd-gui:** java decompiler, uses dex2jar
  - **ded:** dalvik decompiler

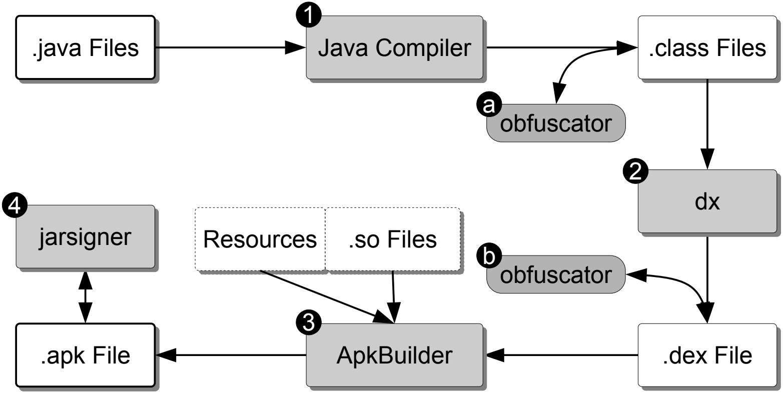
easy to confuse and break ;)

Obfuscation techniques

Obfuscation techniques

- String obfuscation
- Identifier mangling
- Dynamic code loading
- Junkbyte insertion
- Self modifying code

Build process



Dalvik Design

Android Applications are

- ... written in Java
- ... highlevel bytecode
- ... clear defined bytecode
- ... strict model assumptions (verifier)

So should be easy to analyze ...

Dalvik Design

Android Applications are

- ... written in Java
- ... highlevel bytecode
- ... clear defined bytecode
- ... strict model assumptions (verifier)

So should be easy to analyze ... mostly ;)

my fault

```
adb install test.apk
309 KB/s (13010 bytes in 0.041s)
pkg: /data/local/tmp/test.apk
Failure [INSTALL_FAILED_DEXOPT]
```

- checksum?
- wrong method size?
- wrong "goto" destination?
- unsorted string list
- unknown opcode?

Junk Byte

dexdump:

```
0003ac:                        |[0003ac] com.junkbyte.JunkByteActivity.calc:()I
0003bc: 1250                   |0000: const/4 v0, #int 5 // #5
0003be: 3c00 0400              |0001: if-gtz v0, 0005 // +0004
0003c2: 0001 0000 d800 0001    |0003: packed-switch-data (4 units)
0003ca: 0f00                   |0007: return v0
```

androguard:

```
0 0x0 const/4 v0 , [ #+ 5 ] // {5}
1 0x2 if-gtz v0 , [ + 4 ]
2 0x6 nop
3 0x8 nop
4 0xa add-int/lit8 v0 , v0 , [ #+ 1 ]
5 0xe return v0
```

dex2jar + jd-gui

```
    if (5 <= 0);
    return 5;
```

Junk Byte

dexdump:

```
0003ac:                               |[0003ac] com.junkbyte.JunkByteActivity.calc:()I
0003bc: 1250                          |0000: const/4 v0, #int 5 // #5
0003be: 3c00 0400                     |0001: if-gtz v0, 0005 // +0004
0003c2: 1800 0000 d800 0001 0f00 |0003: const-wide v0, #double 0.000000 // #000f01000
```

androguard:

```
0 0x0 const/4 v0 , [ #+ 5 ] // {5}
1 0x2 if-gtz v0 , [ + 4 ]
2 0x6 const-wide v0 , [ #+ 0 ] , [ #+ 216 ] , [ #+ 256 ] , [ #+ 15 ] // {2.0865499802
```

dex2jar + jd-gui

```
if (5 <= 0);
```

new instance

new-instance vAA, type@BBBB

what if class index does not exist?

new instance

- dexdump

  ```
  segmentation fault (core dumped)  dexdump -d test.dex
  ```

- baksmali

  ```
  UNEXPECTED TOP-LEVEL EXCEPTION:
  ```

- androguard

  ```
  IndexError: list index out of range
  ```

- dexdexer

  ```
  java.lang.ArrayIndexOutOfBoundsException: 16896
  ```

Verifier

It's not installable on Android Devices due to verifier rejects such APKs.

not a security feature, but optimization

so, let's mark it as optimized ;)

Verifier

It’s not installable on Android Devices due to verifier rejects such APKs.

not a security feature, but optimization

so, let’s mark it as optimized ;)

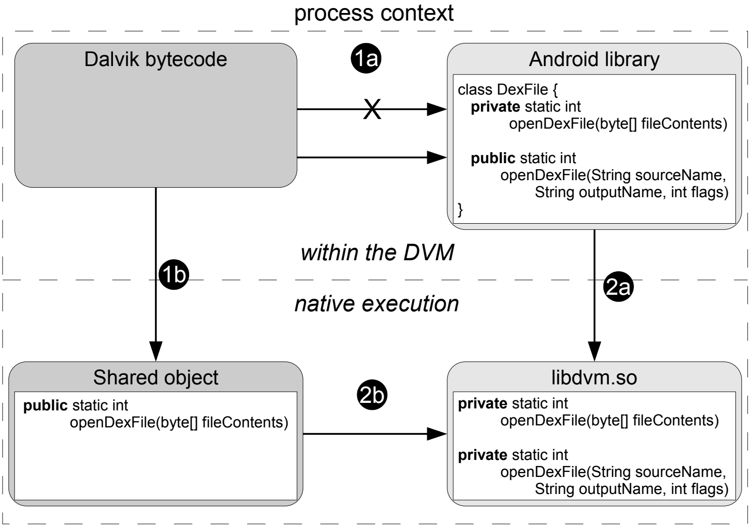
Ups, verifier bug -> device won’t boot

Crypto Loader

DexFile Class enables reflection

- operates on files
- generates and stores optimized dex files
- cool functions are private :(

Crypto Loader



```
class DexFile {
  private static int
        openDexFile(byte[] fileContents)

  public static int
      openDexFile(String sourceName,
            String outputName, int flags)
}
```

```
public static int
      openDexFile(byte[] fileContents)
```

```
private static int
      openDexFile(byte[] fileContents)

private static int
      openDexFile(String sourceName,
            String outputName, int flags)
```

Conclusion

Conclusion

- bytecode constrains are nice, but the verifier implementation has bugs
- packer/dropper can be implemented
- disassemblers have still bugs

Questions?

Thank you for your attention.

email	thuxnder@dexlabs.org
twitter	@thuxnder