Abusing Dalvik Beyond Recognition

Jurriaan Bremer

Who?

Jurriaan Bremer

- Freelance Security Researcher
- Student (University of Amsterdam)
- Interested in Mobile Security & Low-level stuff
  - Core Developer of Cuckoo Sandbox (http://cuckoosandbox.org/)
  - Author of Open Source ARMv7 Disassembler (http://darm.re/)
  - Blog (http://jbremer.org/)
- Eindbazen CTF Team, The Honeynet Project

What?

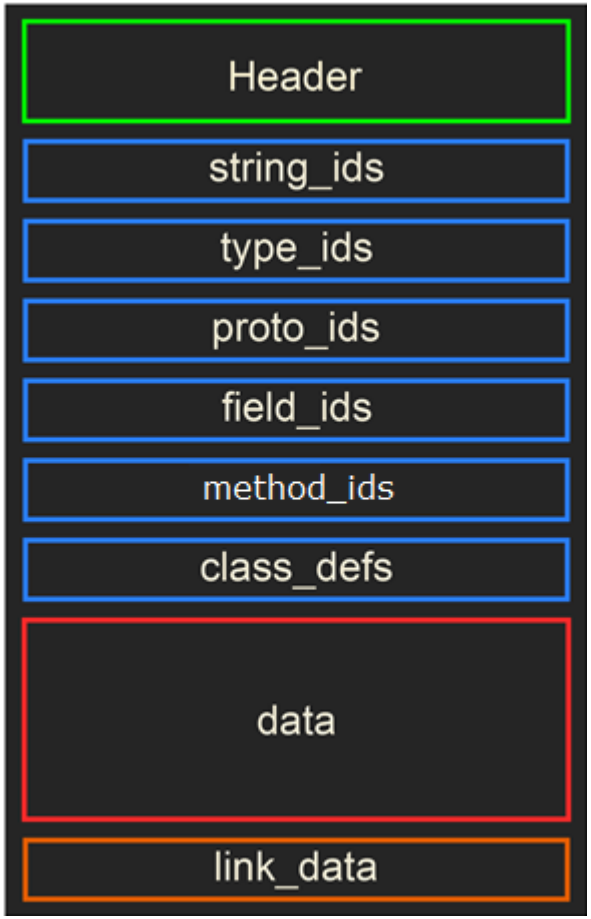
Why?

- Broken stuff is good stuff
- New ways to mess with analysis
- Break analysis tools
- To have fun.. ☺

Android Introduction

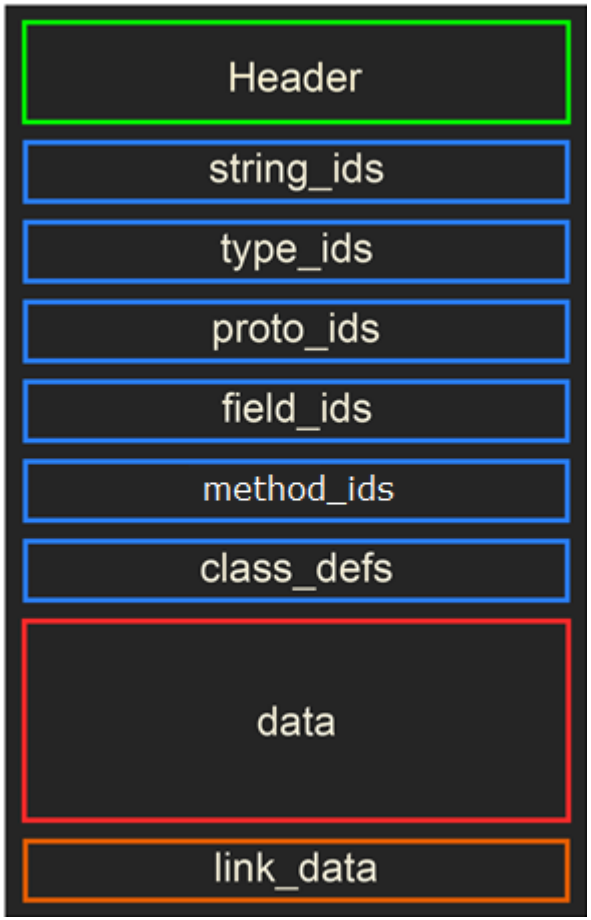
- Android phones (usually) run ARMv7
- Based on a heavily modified Linux kernel
- An application is an APK – a Zip file
  - Contains metadata: signatures, android manifest, etc
  - Code, Images, Data, ..
- Applications' code
  - Mainly written in **Java**, but may contain **native** cod
  - **Dalvik**: Android's Java Virtual Machine
  - All code goes in to **classes.dex** (the Dex file format)

Dex File Format



- Simple File Header
- Various Data Pools
  - Compact Data Structures
    - Fixed-length lookup tables
  - Represent **one** thing each
    - Strings, Data Types
    - Field/Method definition
- Data section
  - Variable-length information
  - E.g., the actual Dalvik code

Dex File Format: Strings



string_id_item = in string_id pool

string_id_item:	
string_data_off	uint

string_data_item = in data section

string_data_item:	
utf16_size	ULEB128
data	ubyte[utf16_size]

ULEB128: Compact storage for small 32-bit ints

Utilizes 1 up to 5 bytes:

- 42 1 byte (0x2a)
- 1337 2 bytes (0xb9 0x0a)
- 0xffffffff 5 bytes (0xff 0xff 0xff 0xff 0x0f)

DexOpt

Strict **verifier** of the Dex File Format

- Enforces a lot of rules
  - See the Dex specification

(http://source.android.com/devices/tech/dalvik/dex-format.html)

Both documented & undocumented
E.g., manual states map_list is optional – it's not.

Manual:

map_off	uint	offset from the start of the file to the map item, or `0` if this file has no map. The offset, if non-zero, should be to an offset into the `data` section, and the data should be in the format specified by "`map_list`" below.

libdex:

```
if (okay) {
    /*
     * Look for the map. Swap it and then use it to find and swap
     * everything else.
     */
    if (pHeader->mapOff != 0) {
        [..]
    } else {
        ALOGE("ERROR: No map found; impossible to byte-swap and verify");
        okay = false;
    }
}
```

DexOpt

Many strict rules, including, e.g.:

- No more padding than required
  - Extra byte of padding? Shame on you!
- Padding must consist of zeroes only
- Entries in the Data Pools must be **unique**
  - May not define the same string twice
- Entries in the Data Pools must be **sorted**
  - string “a” comes before string “b”
  - type 42 comes before type 1337

Dalvik 101

```
public static void hello() {
    System.out.println(“Hello Hack.lu”);
}
```

```
sget-object v0, System;->out:PrintStream;
const-string v1, ”Hello Hack.lu”
invoke-virtual v0, v1, PrintStream;->println(String;)V
return-void
```

Dalvik 102

- Register-based Instruction Set
  - Allocates a fixed-size amount of registers for a function
  - More efficient than Java's stack-based instruction set
- Various General Purpose Instructions
  - Move, add, subtract, multiply, etc
- Fixed branches
  - No "jump register", only "goto $+30" and alike
- Class, Static and Array get/put instructions
  - To read/write class members & array indices
- Special: Switch/case, array-length, const-string, ..

DexOpt Continued

Strict verification of **Dalvik Bytecode**

- All **branches** must point to valid Bytecode
  - Checks for out-of-bounds code access
- **Type checking**
  - Objects can't do arithmetic
  - Strings can't perform the "array-length" instruction
  - Can't "invoke-static" a virtual method
  - Argument count & types must match prototypes
    - E.g., prototype (Lfoo;II)V *requires* 3 parameters

      (One *foo* object and two integers – method has no return value.)

“Parser Differentials”

- Term coined by *Meredith Patterson, Len Sassaman, Sergey Bratus* et al
  - N parsers with 1 input, 1..N different interpretations
  - Parser/Docs inconsistency leads to “funny” stuff
- map_list is a Parser Differential
  - Not a very interesting one though..
  - Hint hint.. ;-)

Straight from the Documentation

access_flags Definitions

embedded in class_def_item, encoded_field, encoded_method, and InnerClass

Bitfields of these flags are used to indicate the accessibility and overall properties of classes and class members.

Name	Value	For Classes (and `InnerClass` annotations)	For Fields	For Methods
ACC_PUBLIC	0x1	`public`: visible everywhere	`public`: visible everywhere	`public`: visible everywhere
ACC_PRIVATE	0x2	* `private`: only visible to defining class	`private`: only visible to defining class	`private`: only visible to defining class
[..]				
(unused)	0x8000			
ACC_CONSTRUCTOR	0x10000			constructor method (class or instance initializer)
ACC_DECLARED_SYNCHRONIZED	0x20000			declared `synchronized`. **Note:** This has no effect on execution (other than in reflection of this flag, per se).

“Parser Diff..**WAIT WHAT?!?!**”

libdex/DexFile.h:

```
 ACC_ENUM          = 0x00004000,       // class, field, ic (1.5)
 ACC_CONSTRUCTOR   = 0x00010000,       // method (Dalvik only)
 ACC_DECLARED_SYNCHRONIZED =
                     0x00020000,       // method (Dalvik only)
 ACC_CLASS_MASK =
     (ACC_PUBLIC | ACC_FINAL | ACC_INTERFACE | ACC_ABSTRACT
                 | ACC_SYNTHETIC | ACC_ANNOTATION | ACC_ENUM),
```

oo/Object.h:

```
    /* unlike the others, these can be present in the optimized DEX file */
    CLASS_ISOPTIMIZED          = (1<<17), // class may contain opt instrs
    CLASS_ISPREVERIFIED        = (1<<16), // class has been pre-verified
};

/* bits we can reasonably expect to see set in a DEX access flags field */
#define EXPECTED_FILE_FLAGS \
    (ACC_CLASS_MASK | CLASS_ISPREVERIFIED | CLASS_ISOPTIMIZED)
```

Dex vs ODex

- ODex – Optimized Dex Files
  - Created after verifying Dex file
  - Various optimizations (CPU-wise)
- **Our Dex is not an ODex file**
  - **CLASS_ISOPTIMIZED|CLASS_ISPREVERIFIED**
  - Well, thanks, eh?
- libdex doesn't verify Dex vs ODex
  - To be continued.

Now what?

We can mark a class “verified & optimized”

- DexOpt will then.. set a status field:

```
/*
 * Done!
 */
if (IS_CLASS_FLAG_SET(clazz, CLASS_ISPREVERIFIED))
    clazz->status = CLASS_VERIFIED;
else
    clazz->status = CLASS_RESOLVED;
okay = true;
```

- Followed by a check:

```
/*
 * If the class hasn't been verified yet, do so now.
 */
if (clazz->status < CLASS_VERIFIED) {
    /*
```

Abuse ALL the Dalvik

- We can now write not-so-strict Dalvik
  - For all methods of a particular class
  - No verification ☺
  - Just set the class' **access_flags**
- Possibilities in Dalvik
  - Write "special" sequences of instructions
    - Normally rejected during validation
  - Use instructions available for ODex
    - Optimized instructions

Goal: Run arbitrary Dalvik

- Input: Raw Dalvik Bytecode
  - Most Dalvik instructions take {1..5} ushort's
  - Use a string with unicode "characters" (Bytecode)
    - Each character represented as UTF-16 "code point"
    - UTF-16 code points are 16-bits – like an ushort
- Task: Redirect Dalvik's Program Counter
  - To the string with our Bytecode
- Output: The return value
  - After executing our raw Dalvik Bytecode ☺

Some Gadgets

We're going to require some basic stuff

- Object **address leak**
  - *What is the address of our Object?*
- **Read** arbitrary integer
  - *What is the value at this address?*
- **Write** arbitrary integer
  - Your address now contains my value! ☺

Gadgets: Object Address Leak

Can simply **cast** an Object as **integer**

(Now Type Checking is disabled ☺)

```
// Invalid Java code, but closest estimation
// to our Bytecode
int address(Object obj) {
    return (int) obj;
}
```

Gadgets: Read Arbitrary Integer

- We use the "**array-length**" instruction
  - Arrays, e.g., *int[] foo = new int[42];*
  - Arrays in Dalvik have their length at offset **+8**
- Our read_int32 function
  - Subtract 8 from the address
  - Perform "array-length" on our address
  - Return the "length"

Gadgets: Write Arbitrary Integer

- Usage of "**iput-quick**" instruction
  - **iput** = Instance Put, set a field of an instance object
  - E.g., *this.foo = bar;*
  - **v0** = bar, **v1** = this →
  - *iput v0, v1, SomeClass;->foo:I*
- **Quick** is the ODex version
  - *iput-quick v0, v1, **#+4***
  - **#+4** is the offset of field **foo** from **this**
  - Can overwrite any "field" with **iput-quick**

Strings in Java

- String is a wrapper around char[]
  - *(u32 *)(str + 8) = pointer to char[]
  - (u16 *)(char[] + 16) = UTF-16 code points
- E.g., given string “Hack.lu \u1337”
  - UTF-16 code points will look like:

```
$ py -c 'print u"Hack.lu \u1337".encode("utf-16le")'|xxd
0000000: 4800 6100 6300 6b00 2e00 6c00 7500 2000  H.a.c.k...l.u. .
0000010: 3713                                     7.
```

Executing Arbitrary Dalvik

- We want to execute our Dalvik String
- Override the address of a **virtual function**
- Class layout:
  - *(u32 *)(**this** + 0) = **clazz** object
  - *(u32 *)(**clazz** + 112) = vtable_count
  - *(u32 *)(**clazz** + 116) = vtable_pointer
- All classes inherit *java.lang.Object*
  - *Which defines a couple of virtual methods itself*
- We create a custom class with **1 virtual method**
  - Our virtual method is located at index **vtable_count-1**

Executing Arbitrary Dalvik

- vtable: pointers to **Method** instances
- vm/mterp/armv5te/footer.S:

```
.LinvokeArgsDone: @ r0=methodToCall
    ldrh    r9, [r0, #offMethod_registersSize]  @ r9<- methodToCall->regsSize
    ldrh    r3, [r0, #offMethod_outsSize]  @ r3<- methodToCall->outsSize
    ldr     r2, [r0, #offMethod_insns]  @ r2<- method->insns
```

- vm/mterp/common/asm-constants.h:

```
MTERP_OFFSET(offMethod_registersSize,   Method, registersSize, 10)
MTERP_OFFSET(offMethod_outsSize,        Method, outsSize, 12)
MTERP_OFFSET(offMethod_name,            Method, name, 16)
MTERP_OFFSET(offMethod_insns,           Method, insns, 32)
```

- Pointer to Dalvik Bytecode at offset 32

Quick Pwn Summary

- Get an arbitrary String
  - Locate its UTF-16 code points (our Bytecode)
- Create Object of a Class with a virtual method
  - Get last vtable entry
  - Overwrite Insns with the address to our Bytecode
- Call the virtual method:
  - v0 = object instance
  - *invoke-virtual {v0}, SomeClass;->dummy_method*

Demo o'clock

- Our Bytecode should return gracefully
  - (It's too easy to crash the emulator at this point..)
  - We can even get its return value ☺
- Made a simple Application
  - With a textbox, waiting for Bytecode
  - A fancy button
  - Shows the return value of the executed Bytecode
    - Represented as integer below the button

Demo Time ☺

Bytecode Examples

```
$ py dalvik.py '0013 0539 000f'
0     const/16 v0, #0x539
2     return v0

$ py dalvik.py '0013 0539 00d8 0300 000f'
0     const/16 v0, #0x539
2     add-int/lit8 v0, v0, #+3
4     return v0
```

Real usage?

- We can put any Bytecode we want
  - Including invalid Bytecode (just don't invoke it)
  - Breaks commonly used tools, big time
    - Exercise for the reader
- We can run arbitrary Dalvik Bytecode
  - No need to hardcode all our proprietary code
  - Prevent easy analysis of your Application
    - Because decompiling "normal" Dalvik into Java is damn easy

Future Work

Native Code Execution

- (Directly from within Dalvik, naturally)
- Definitely possible, but requires some work..
- Need to allocate RWX memory **or** use ROP
  - Will probably want to parse /proc/self/maps
  - Locate **mmap()** or **mprotect()**
- Set **ACC_STATIC** in **access_flags** for virtual method
  - Allows to jump to arbitrary ARMv7 code ☺

Future Work

- Self-decrypting Dalvik Bytecode
  - Don't run the entire Dalvik string right away
  - Pass only chunks – mutate parts on-the-go
  - Whatever you can think of..?
- Obfuscate the memory corruption gadgets
  - Right now it's pretty obvious..
- Exploit other built-in classes & features
- **Modify the Dalvik VM itself**
  - Facebook "extended" the Dalvik VM for >64k methods

    (*invoke*-* instructions normally take a 16-bit index.)

For fun: execute-inline

- Optimizations of a few dozen functions, e.g.:

```
const InlineOperation gDvmInlineOpsTable[] = {
    { org_apache_harmony_dalvik_NativeTestTarget_emptyInlineMethod,
        "Lorg/apache/harmony/dalvik/NativeTestTarget;",
        "emptyInlineMethod", "()V" },

    { javaLangString_charAt, "Ljava/lang/String;", "charAt", "(I)C" },
    { javaLangString_compareTo, "Ljava/lang/String;", "compareTo", "(Ljava/lang/String;)I" },
    { javaLangString_equals, "Ljava/lang/String;", "equals", "(Ljava/lang/Object;)Z" },
    { javaLangString_fastIndexOf_II, "Ljava/lang/String;", "fastIndexOf", "(II)I" },
    { javaLangString_isEmpty, "Ljava/lang/String;", "isEmpty", "()Z" },
    { javaLangString_length, "Ljava/lang/String;", "length", "()I" },
```

- *execute-inline {v0, v1, v2}, 42@inline*
- Doesn't do bounds checking
- Table is close to GOT
  - Exposes some functions, e.g., **memcpy**, **mmap** :p

For Fun: invoke-super-quick

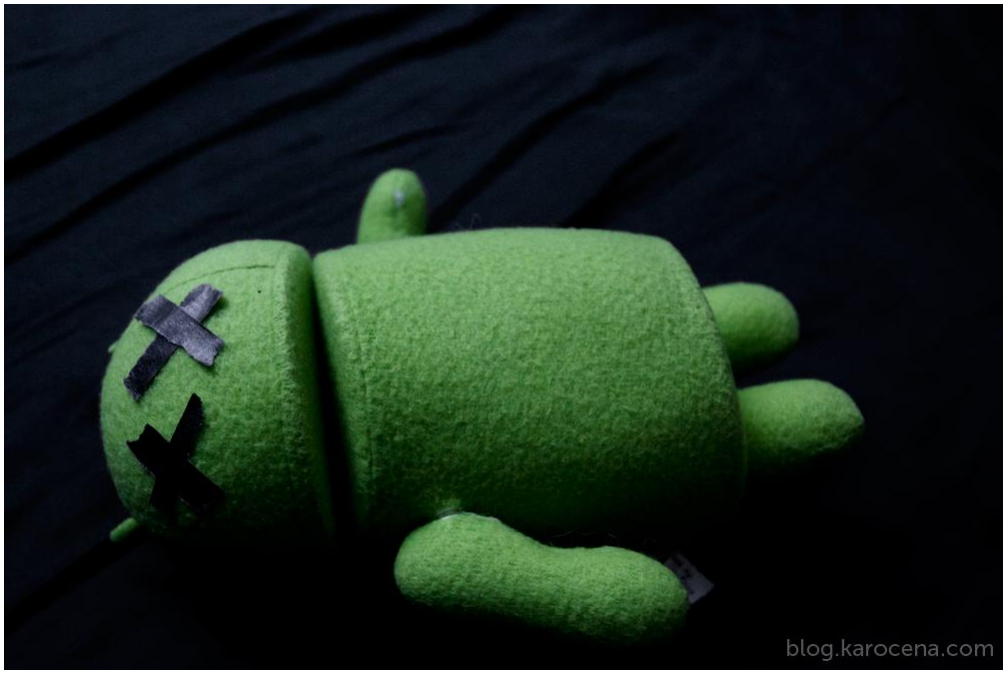
- Invokes the *super* method for a virtual method
- Takes a bit more time to setup
  - Create a class **A** with a virtual method
  - Create a class **B** which inherits class **A**
  - Overwrite Insns address for **A**'s virtual method
  - Call **A**'s virtual method from **B**'s with *super*
- More awesome ☺
  - Doesn't invoke a *virtual* method
  - Invokes a ***super quick*** method ☺☺

Patch by Ben Gruver (JesusFreke) (PoC still works on Android 4.3?!)

```
$ git show c2e9a5b2b70d69c
commit c2e9a5b2b70d69c027964c9a4d07a4bdf723dd36
Author: Ben Gruver <bgruv@google.com>
Date:   Wed May 8 13:29:36 2013 -0700

    Move verification of class access flags to libdex

    Change-Id: I020a168cfff46e319b0bebb6c7477f0b4139c6de

diff --git a/libdex/DexSwapVerify.cpp b/libdex/DexSwapVerify.cpp
index 24a86f9..2bb403e 100644
--- a/libdex/DexSwapVerify.cpp
+++ b/libdex/DexSwapVerify.cpp
@@ -911,6 +911,11 @@ static void* swapClassDefItem(const CheckState* state, void* ptr) {
     SWAP_OFFSET4(item->annotationsOff);
     SWAP_OFFSET4(item->classDataOff);

+    if ((item->accessFlags & ~ACC_CLASS_MASK) != 0) {
+        ALOGE("Bogus class access flags %x", item->accessFlags);
+        return NULL;
+    }
+
     return item + 1;
 }
```

The End.

The Real End ☺



Jurriaan Bremer
me@jbremer.org
@skier_t

Thanks to: Alexandre Dulaunoy, Patrick Schulz, Rodrigo Chiossi, Sergey Bratus, Valentin Pistol, ShiftReduce, Thomas Schreck, Peter Geissler, Eindbazen CTF Team