Reconstructing Dalvik applications

Marc Schönefeld

University of Bamberg

HITB Dubai 2010

Agenda

Introduction

Dalvik development from a RE perspective

Parsing Strategy

Processing the results

Dealing with optimization

Finalizing

The Speaker

Marc Schönefeld

- since 2002 Talks on Java-Security on intl. conferences (Blackhat, RSA, DIMVA,PacSec, CanSecWest, SyScan, HackInTheBox)
- day time busy for Red Hat (since 2007)
- PhD Student at University of Bamberg (defended January 2010)

Motivation

- As a reverse engineer I have the tendency to look in the code that is running on my mobile device
- Coming from a JVM background I wanted to know what Dalvik is really about
- Wanted to learn some yet another bytecode language
- Coding is fun, especially when dragged into bureaucratic actions otherwise

What is Dalvik

- Dalvik is the runtime that runs userspace Android applications
- invented by Dan Bornstein (Google)
- named after a village in Iceland
- register-based
- runs own bytecode dialect, which is very similar but not equal to java bytecode

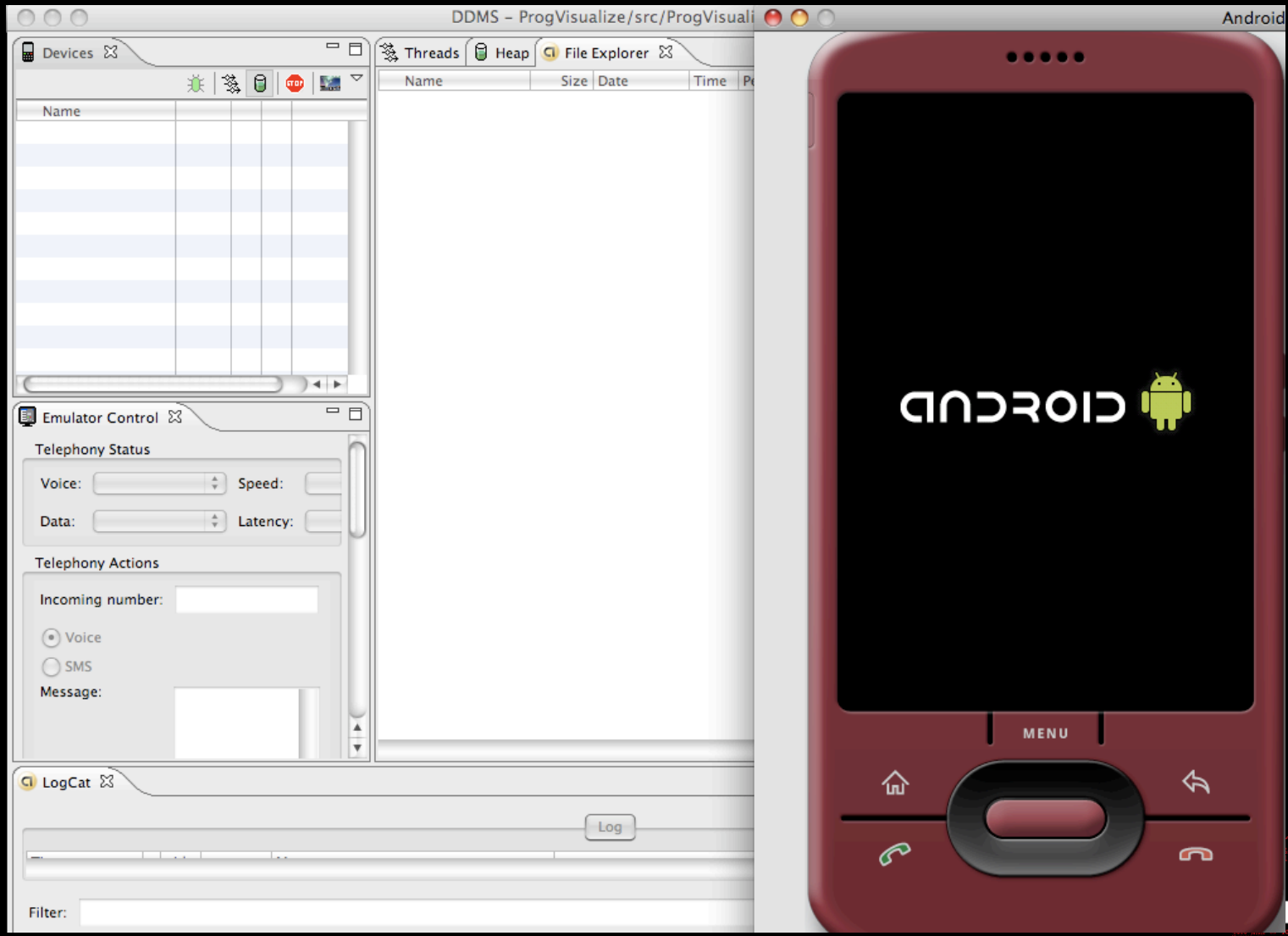
Dalvik vs. JVM

	Dalvik	JVM
Architecture	Register	Stack
OS-Support	Android	Multiple
RE-Tools	few	many
Executables	APK	JAR
Constant-Pool	per Application	per Class

Dalvik Development environment

- Dalvik apps are developed using java developer tools on a standard desktop system (like eclipse),
- compiled to java classes (javac)
- transformed to DX with the dx tool (classes.dex)
- classes.dex plus meta data and resources go into a dalvik application 'apk' container
- this is transferred to the device or an emulator (adb, or download from android market)

Dalvik Development process



Dalvik runtime libraries

- You find the core classes that are available to build Dalvik applications in the `android.jar` file.
- These are stub classes to fulfil dependencies during linking

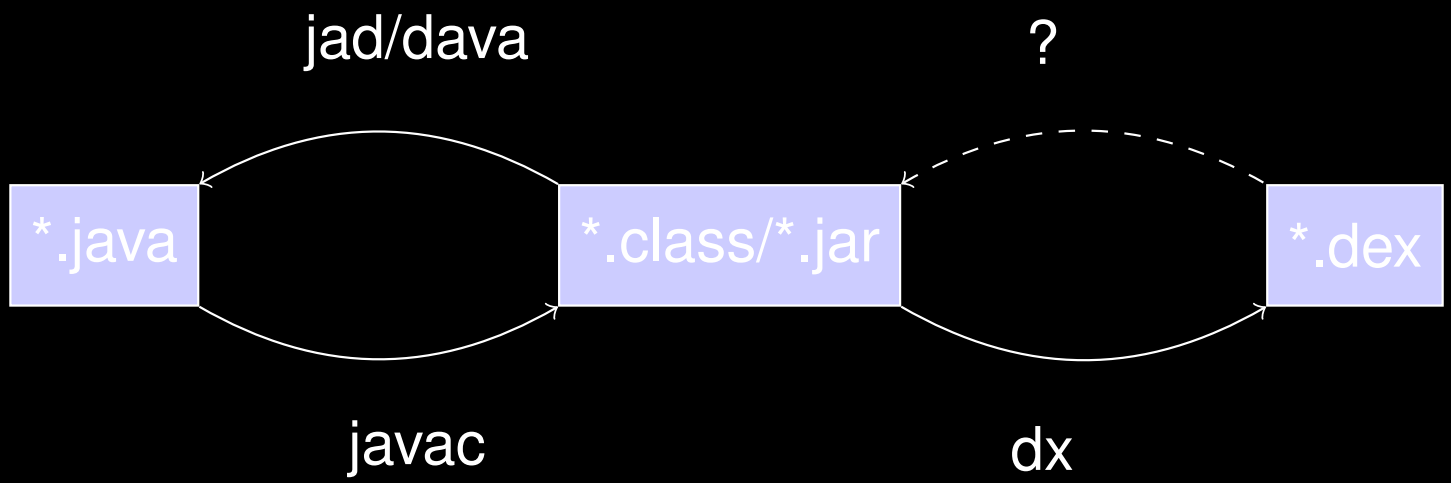
	Dalvik	JVM
java.io.*	Y	Y
java.net.*	Y	Y
android.*	Y	N
dalvik.*	Y	N
com.google.*	Y	N
javax.swing.*	N	Y
...	...	...

Dalvik development from a RE perspective

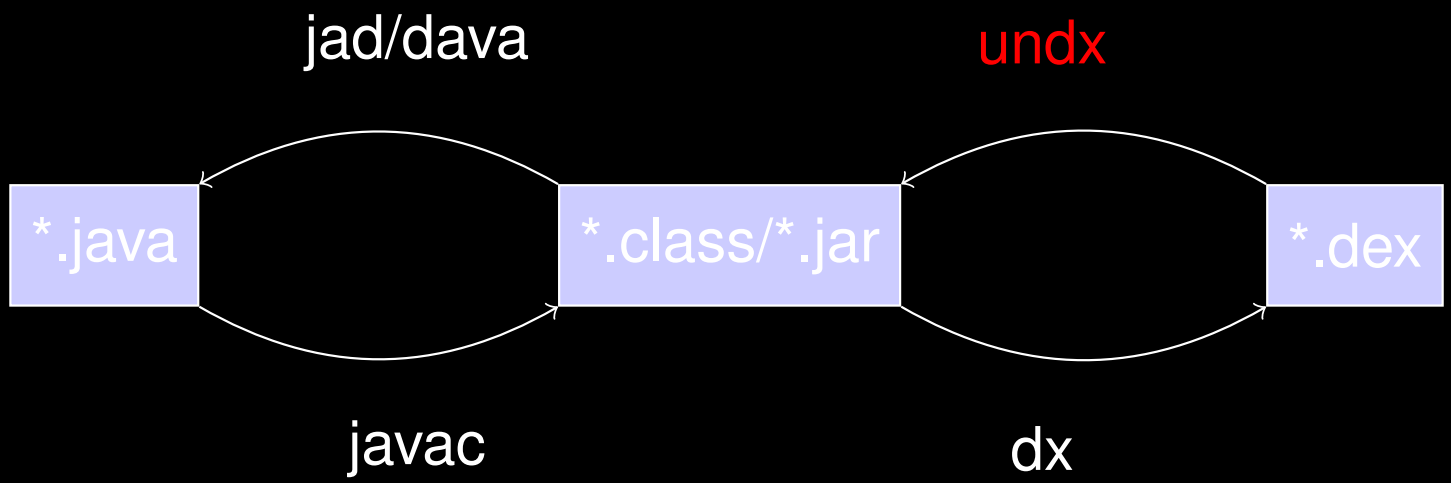
Dalvik applications are available as Android Packages (*.apk), no source included, so you buy/download a cat in the bag.
How can you find out, whether

- the application contains malicious code, ad/spyware, or phones home to the vendor ?
- has unpatched security holes (dex generated from vulnerable java code) ?
- contains copied code, which may violate GPL or other license agreements ?

Dalvik development from a RE perspective



Filling the gap



Tool design choices

- How to parse dex files?
  - write a complicated DEX parser
  - or utilize something existing
- How to translate to class files (bytecode library)?
  - ASM
  - BCEL

Parsing DEX files

- The dexdump tool of the android SDK can perform a complete dump of dex files, it is used by undx

	dexdump	parsing directly
Speed	Time advantage, do not have to write everything from	Direct access to binary structures (arrays, jump tables)
Control	dexdump has a number of nasty bugs	Immediate fix possible
Available info	Filters a lot	All you can parse
...	...	...

- The decision was to use as much of useable information from dexdump, for the rest we parse the dex file directly
- The use of ddx or smali was evaluated, but those libraries not reduce with the custom parsing effort

Parsing DEX files

- This is useful dexdump output, good to parse

```
    #1              : (in LUnDxTest;)
      name          : 'main'
      type          : '([Ljava/lang/String;)V'
      access        : 0x0009 (PUBLIC STATIC)
      code          -
      registers     : 3
      ins           : 1
      outs          : 2
      insns size    : 20 16-bit code units
00024c:                                        |[00024c] UnDxTest.main:([Ljava/lang/String;)V
00025c: 2200 0200                              |0000: new-instance v0, LObj; // class@0002
000260: 7010 0000 0000                         |0002: invoke-direct {v0}, LObj;.<init>:()V // method@0000
000266: 1251                                   |0005: const/4 v1, #int 5 // #5
000268: 6e20 0200 1000                         |0006: invoke-virtual {v0, v1}, LObj;.doit:(I)V // method@0002
00026e: 1301 ff00                              |0009: const/16 v1, #int 255 // #ff
000272: 6e20 0200 1000                         |000b: invoke-virtual {v0, v1}, LObj;.doit:(I)V // method@0002
000278: 1301 f000                              |000e: const/16 v1, #int 240 // #f0
00027c: 6e20 0200 1000                         |0010: invoke-virtual {v0, v1}, LObj;.doit:(I)V // method@0002
000282: 0e00                                   |0013: return-void
```

Parsing DEX files

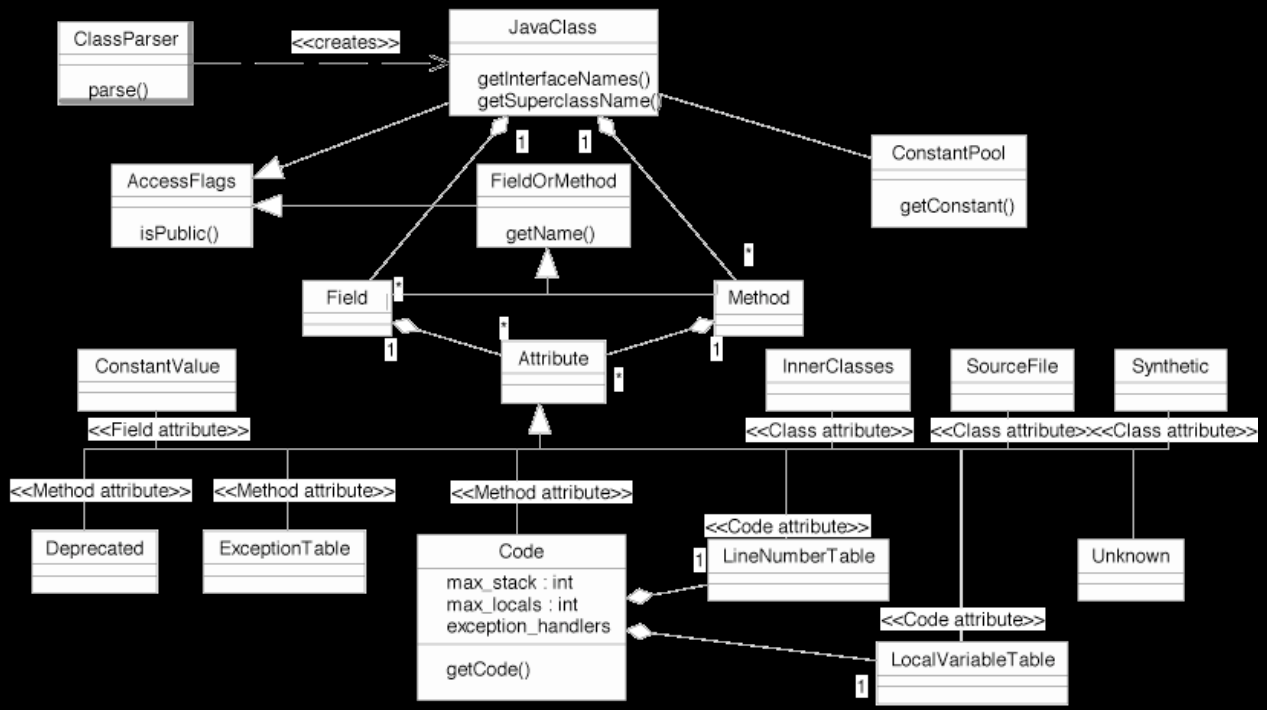
- This is useful dexdump output, omitting important data

```
name          : '<clinit>'
type          : '()V'
access        : 0x10008 (STATIC CONSTRUCTOR)
code          -
registers     : 1
ins           : 0
outs          : 0
insns size    : 34 16-bit code units
000310:                                        |[000310] MD5.<clinit>:()V
000320: 1200                                   |0000: const/4 v0, #int 0 // #0
000322: 6900 0200                              |0001: sput-object v0, LMD5;.md5:LMD5; // field@0002
000326: 1300 1000                              |0003: const/16 v0, #int 16 // #10
00032a: 2300 0f00                              |0005: new-array v0, v0, [C // class@000f
00032e: 2600 0700 0000                         |0007: fill-array-data v0, 0000000e // +00000007
000334: 6900 0000                              |000a: sput-object v0, LMD5;.hexChars:[C // field@0000
000338: 0e00                                   |000c: return-void
00033a: 0000                                   |000d: nop // spacer
00033c: 0003 0200 1000 0000 3000 3100 3200 ... |000e: array-data (20 units)
catches       : (none)
positions     :
0x0000 line=7
0x0003 line=8
```

Strategy

- extract classes.dex from *.apk file
- parse global structures (constants)
- for each class in dex
  - parse class meta data
  - for each method in class
    - Parse method meta data
    - for each instructions in method
      - transform to java bytecode
    - generate java method (BCEL method)
  - generate java class (BCEL method)
  - store class in jar

- http://jakarta.apache.org/bcel/
- We chose the BCEL library from Apache as it has a very broad functionality (compared to alternatives like ASM and javassist)



Structure

- **extract classes.dex from *.apk file**
- **parse global structures (constants)**
- for each class in dex
  - parse class meta data
  - for each method in class
    - Parse method meta data
    - for each instructions in method
      - transform to java bytecode
    - generate java method (BCEL method)
  - generate java class (BCEL method)
  - store class in jar

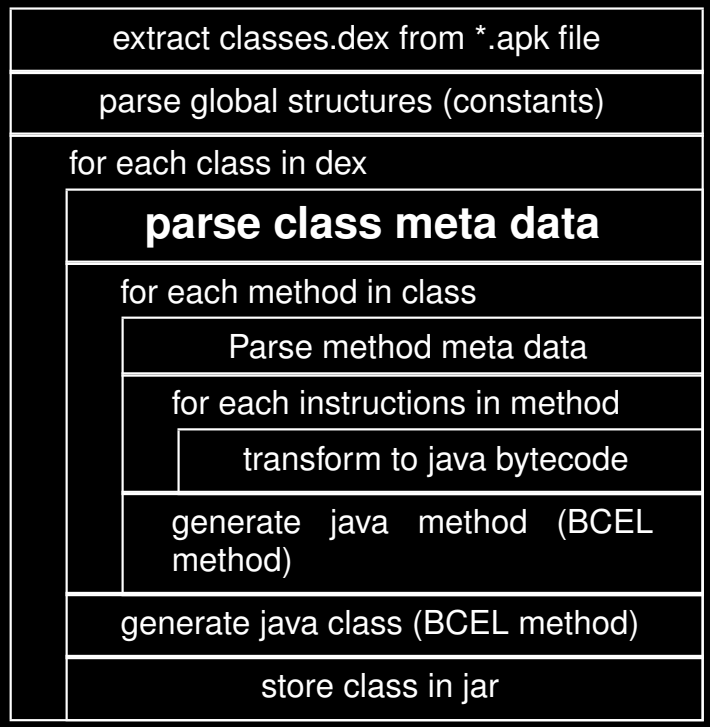
- Extract global meta information
- Transform into relevant BCEL constant structures
- Retrieve the string table to prepare the Java constant pool

Process classes

- extract classes.dex from *.apk file
- parse global structures (constants)
- **for each class in dex**
  - parse class meta data
  - for each method in class
    - Parse method meta data
    - for each instructions in method
      - transform to java bytecode
    - generate java method (BCEL method)
  - generate java class (BCEL method)
  - store class in jar

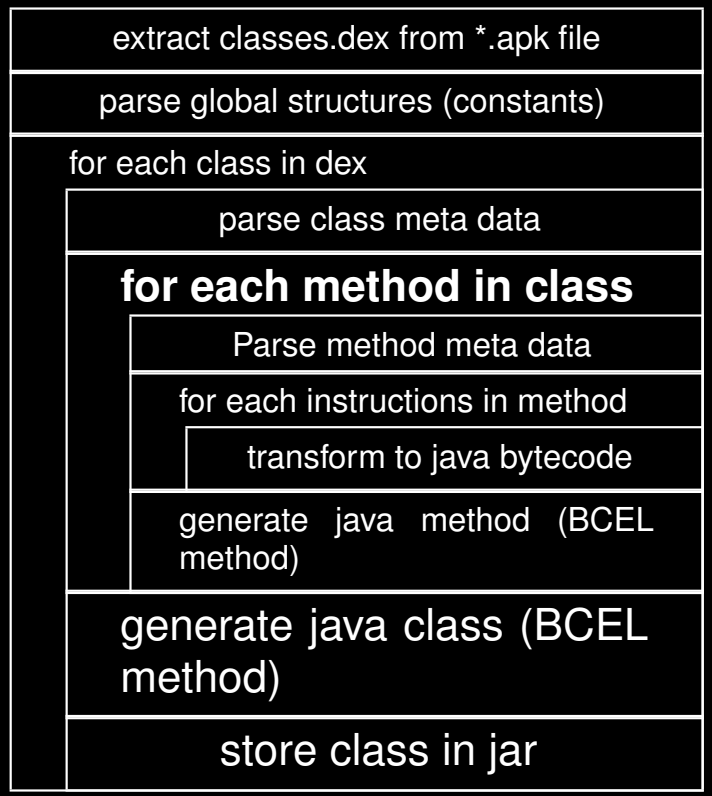
- Transform each class
- Parse Meta Data
- Process methods
- Generate BCEL class
- Dump class file

Process class Meta Data



- Extract Class Meta Data
- Visibility, class/interface, classname, subclass
- Transfer static and instance fields

Process the individual methods



- Extract Method Meta Data
- Parse Instructions
- Generate JAVA method

Parse Method Meta Data

- extract classes.dex from *.apk file
- parse global structures (constants)
- for each class in dex
  - parse class meta data
  - for each method in class
    - **Parse method meta data**
    - for each instructions in method
      - transform to java bytecode
    - generate java method (BCEL method)
  - generate java class (BCEL method)
  - store class in jar

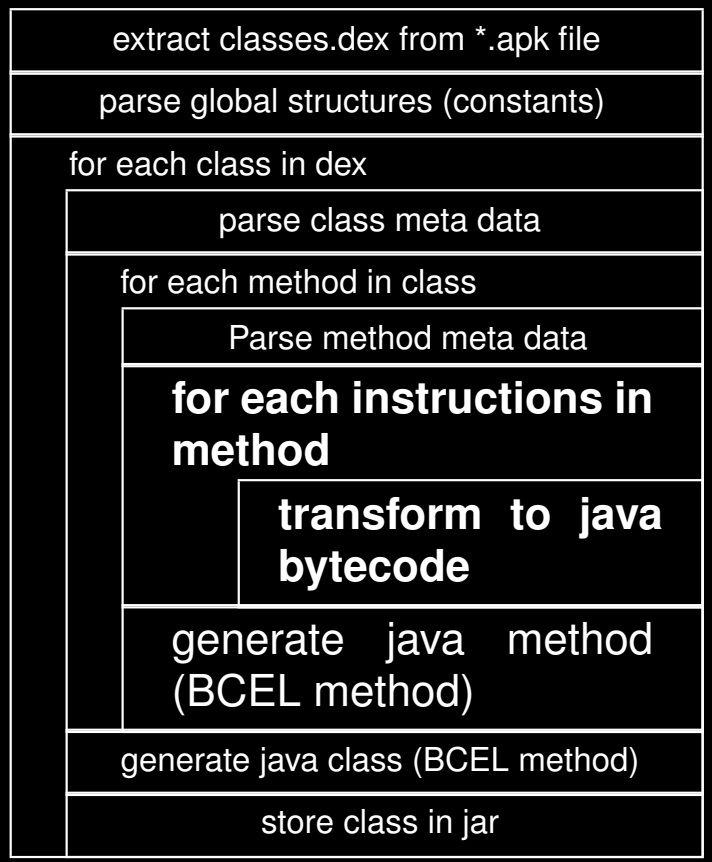
- transform method meta data to BCEL method structures
- extract method signatures,
- set up local variable tables,
- map Dalvik registers to JVM registers

Acquire method meta data

Acquire method meta data

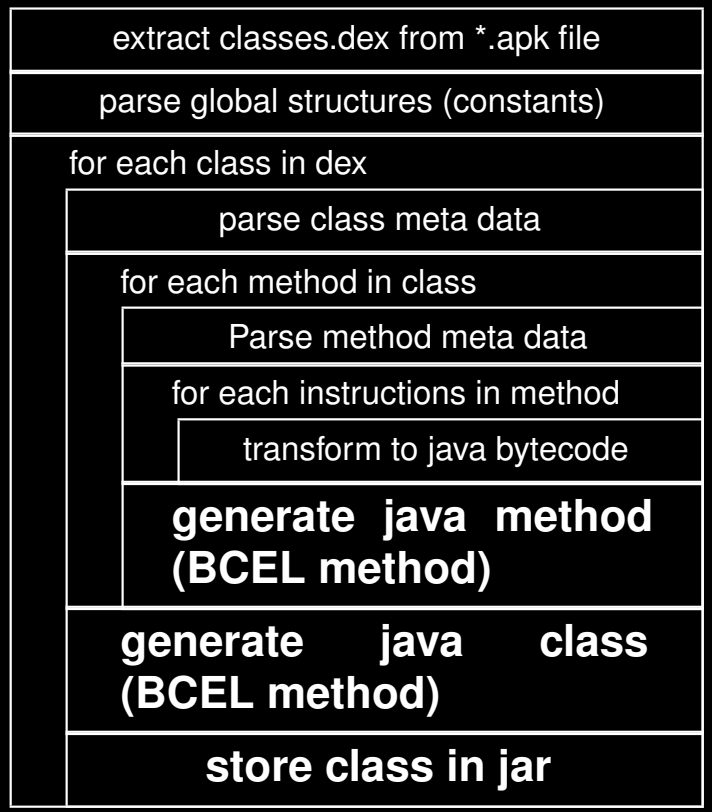
```
private MethodGen getMethodMeta(ArrayList<String> al, ConstantPoolGen pg,
                        String classname) {

for (String line : al) {
        KeyValue kv = new KeyValue(line.trim());
        String key = kv.getKey(); String value = kv.getValue();
        if (key.equals(str_TYPE)) type = value.replaceAll("'", "");

        if (key.equals("name")) name = value.replaceAll("'", "");

        if (key.equals("access")) access = value.split(" ")[0].substring(2);

        allfound = (type.length() * name.length() * access.length() != 0);
        if (allfound) break;
}

Matcher m = methodtypes.matcher(type);

boolean n = m.find();
Type[] rt = Type.getArgumentTypes(type);
Type t = Type.getReturnType(type);

int access2 = Integer.parseInt(access, 16);

MethodGen fg = new MethodGen(access2, t, rt, null, name, classname,
                        new InstructionList(), pg);
return fg;
```

Generate the instructions



- first create BCEL InstructionList
- create NOP proxies for every Dalvik instruction to prepare jump targets (satisfy forward jumps)
- For every Dalvik instruction add an equivalent JVM bytecode block to the JVM InstructionList

Store generated data in BCEL structures



- generate the BCEL structures
- store to current context
- in the end we have a class file for each defined class in the dex

Transforming opcodes

Example: Transforming the new-array opcode

```
private static void handle_new_array(String[] ops, InstructionList il,
                ConstantPoolGen cpg, LocalVarContext lvg) {

String vx = ops[1].replaceAll(",", "");
String size = ops[2].replaceAll(",", "");
String type = ops[3].replaceAll(",", "");
il.append(new ILOAD((short) lvg.didx2jvmidxstr(size)));
if (type.substring(1).startsWith("L")
                || type.substring(1).startsWith("[")) {
         il.append(new ANEWARRAY(Utils.doAddClass(cpg, type.substring(1))));
} else
        {
        il        .append(new NEWARRAY((BasicType) Type.getType(type
                                        .substring(1))));
    }
    il.append(new ASTORE(lvg.didx2jvmidxstr(vx)));
}
```

Transforming opcodes

Example: Transforming virtual method calls

```
private static void handle_invoke_virtual(String[] regs, String[] ops,
          InstructionList il, ConstantPoolGen cpg,
          LocalVarContext lvg,
        OpcodeSequence oc, DalvikCodeLine dcl) {

        String classandmethod = ops[2].replaceAll(",", "");
        String params = getparams(regs);
        String a[] = extractClassAndMethod(classandmethod);

        int metref = cpg.addMethodref(Utils.toJavaName(a[0]), a[1], a[2]);
        genParameterByRegs(il, lvg, regs, a, cpg, metref, true);
        il.append(new INVOKEVIRTUAL(metref));
        DalvikCodeLine nextInstr = dcl.getNext();

        if (!nextInstr._opname.startsWith("move-result")
                && !classandmethod.endsWith(")V")) {
                if (classandmethod.endsWith(")J") ||
                        classandmethod.endsWith(")D")) {
                        il.append(new POP2());
                } else {
                        il.append(new POP());
                }
        }
}
```

Transforming opcodes

Example: Transforming sparse switches

```
String reg = ops[1].replaceAll(",", "");
String reg2 = ops[2].replaceAll(",", "");
DalvikCodeLine dclx = bl1.getByLogicalOffset(reg2);
int phys = dclx.getMemPos();
int curpos = dcl.getPos();
int magic = getAPA().getShort(phys);
if (magic != 0x0200) {        Utils.stopAndDump("wrong magic"); }
int size = getAPA().getShort(phys + 2);
int[] jumpcases = new int[size];
int[] offsets = new int[size];
InstructionHandle[] ihh = new InstructionHandle[size];
for (int k = 0; k < size; k++) {
        jumpcases[k] = getAPA().getShort(phys + 4 + 4 * k);
        offsets[k] = getAPA().getShort(phys + 4 + 4 * (size + k));
        int newoffset = offsets[k] + curpos;
        String zzzz = Utils.getFourCharHexString(newoffset);
        ihh[k] = ic.get(zzzz);
}

int defaultpos = dcl.getNext().getPos();
String zzzz = Utils.getFourCharHexString(defaultpos);
InstructionHandle theDefault = ic.get(zzzz);
il.append(new ILOAD(locals.didx2jvmidxstr(reg)));
LOOKUPSWITCH ih = new LOOKUPSWITCH(jumpcases, ihh, theDefault);
il.append(ih);
```

Store generated data in BCEL structures

Dalvik

```
type          : '()LMD5;'
access        : 0x0009 (PUBLIC STATIC)
code          -
registers     : 1
ins           : 0
outs          : 1
insns size    : 14 16-bit code units
0003c0:                                        |[0003c0] MD5.getInstance:()LMD5;
0003d0: 6200 0200                              |0000: sget-object v0, LMD5;.md5:LMD5; // field@0002
0003d4: 3900 0900                              |0002: if-nez v0, 000b // +0009
0003d8: 2200 0200                              |0004: new-instance v0, LMD5; // class@0002
0003dc: 7010 0100 0000                         |0006: invoke-direct {v0}, LMD5;.<init>:()V // method@0001
0003e2: 6900 0200                              |0009: sput-object v0, LMD5;.md5:LMD5; // field@0002
0003e6: 6200 0200                              |000b: sget-object v0, LMD5;.md5:LMD5; // field@0002
0003ea: 1100                                   |000d: return-object v0
catches       : (none)
positions     :
0x0000 line=24
0x0004 line=26
0x000b line=29
```

JVM code

```
public static MD5 getInstance();
Code:
   0:   getstatic       #14; //Field md5:LMD5;
   3:   astore_0
   4:   aload_0
   5:   ifnonnull       20
   8:   new     #4; //class MD5
   11:  astore_0
   12:  aload_0
   13:  invokespecial   #73; //Method "<init>":()V
   16:  aload_0
   17:  putstatic       #14; //Field md5:LMD5;
   20:  getstatic       #14; //Field md5:LMD5;
   23:  astore_0
   24:  aload_0
   25:  areturn
```

Challenges

- Assign Dalvik regs to jvm regs
- obey stack balance rule (when processing opcodes)
- type inference (reconstruct flow of data assignment opcodes)

Store generated data in BCEL structures

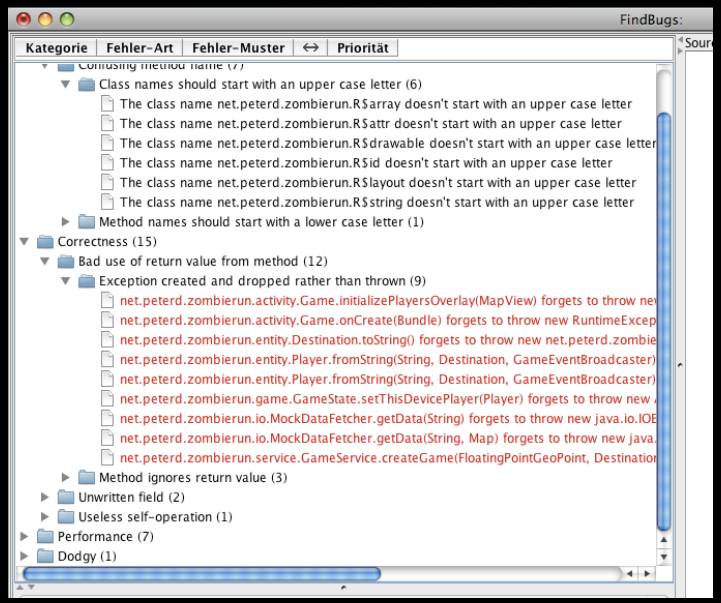
Dalvik

```
type          : '()LMD5;'
access        : 0x0009 (PUBLIC STATIC)
code          -
registers     : 1
ins           : 0
outs          : 1
insns size    : 14 16-bit code units
0003c0:                                        |[0003c0] MD5.getInstance:()LMD5;
0003d0: 6200 0200                              |0000: sget-object v0, LMD5;.md5:LMD5; // field@0002
0003d4: 3900 0900                              |0002: if-nez v0, 000b // +0009
0003d8: 2200 0200                              |0004: new-instance v0, LMD5; // class@0002
0003dc: 7010 0100 0000                         |0006: invoke-direct {v0}, LMD5;.<init>:()V // method@0001
0003e2: 6900 0200                              |0009: sput-object v0, LMD5;.md5:LMD5; // field@0002
0003e6: 6200 0200                              |000b: sget-object v0, LMD5;.md5:LMD5; // field@0002
0003ea: 1100                                   |000d: return-object v0
catches       : (none)
positions     :
0x0000 line=24
0x0004 line=26
0x000b line=29
```

JVM code

```
 public static MD5 getInstance();
Code:
 0:   getstatic       #14; //Field md5:LMD5;
 3:   astore_0
 4:   aload_0
 5:   ifnonnull       20
 8:   new     #4; //class MD5
 11:  astore_0
 12:  aload_0
 13:  invokespecial   #73; //Method "<init>":()V
 16:  aload_0
 17:  putstatic       #14; //Field md5:LMD5;
 20:  getstatic       #14; //Field md5:LMD5;
 23:  astore_0
 24:  aload_0
 25:  areturn
```

Now we have bytecode, what to do with it?



Statically analyze it!

- Analyze the code with static checking tools (findbugs)
- Programming bugs, vulnerabilities, license violations

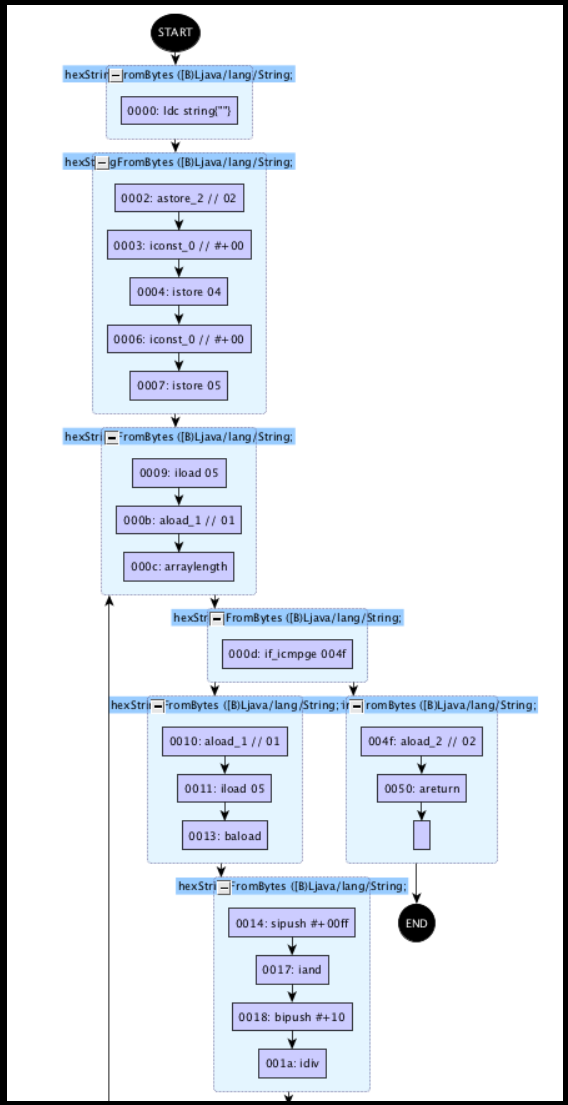
Now we have bytecode, what to do with it?

```
public class WebDialog extends Dialog
{

    public WebDialog(Context arg0)
    {
        super(arg0);
        Object obj = JVM INSTR new #14  <Class WebView>;
        ((WebView) (obj)).WebView(arg0);
        webView = ((WebView) (obj));
        obj = webView;
        obj = ((WebView) (obj)).getSettings();
        boolean flag = true;
        ((WebSettings) (obj)).setJavaScriptEnabled(flag);
        obj = webView;
        setContentView(((android.view.View) (obj)));
        obj = "Welcome";
        setTitle(((CharSequence) (obj)));
    }

    public void loadUrl(String arg0)
    {
        WebView webview = webView;
        webview.loadUrl(arg0);
```

Decompile it!

- Feed the generated jar into a decompiler
- It will spit out JAVA-like code
- Structural equal to the original source (but some differences due to heavy reuse of stack variables)

Now we have bytecode, what to do with it?



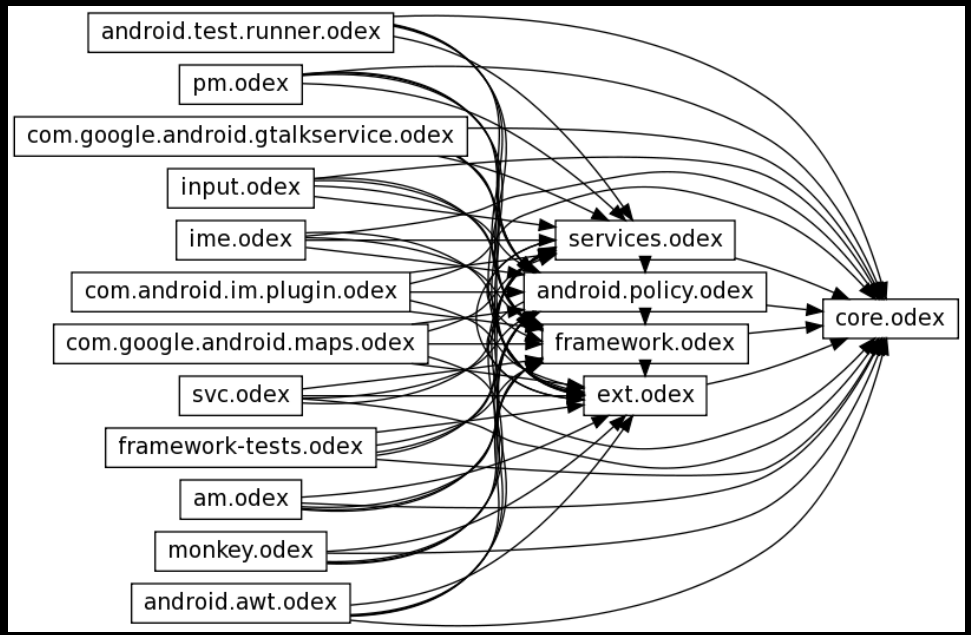
Graph it!

- Script a control-flow-graph analyzer (splitbasic.py included)
- Generate nodes and edges
- Write Graphml file

ODEX dependencies

ODEX file hierarchy

- Over-the-air updates are zipped system images (complete or Delta-patches), with
- applications and libs that come in *.odex files,
- which have chained dependencies



Challenges

Platform-Optimized Dex Files

- ODEX (Optimized Dexfiles) for faster execution on real device platform
  - inlined functions
  - quick-invokes (using vtables)
  - quick field access (using offsets)
- Although undx contains experimental code to handle odex files, using a separate step with smali is more reliable (`code.google.com/p/smali/wiki/DeodexInstructions`)

Inlined Functions

Inline method call

```
08d8fc:                 |[08d8fc] SQLite.JDBC2y.JDBCDatabaseMetaData.
 getCrossReference:(
 Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
 Ljava/lang/String;Ljava/lang/String;
 Ljava/lang/String;)Ljava/sql/ResultSet;
08d90c: 1209            |0000: const/4 v9, #int 0 // #0
08d90e: 3815 4000       |0001: if-eqz v21, 0041 // +0040
08d912: 120d            |0003: const/4 v13, #int 0 // #0
08d914: 0800 1500       |0004: move-object/from16 v0, v21
08d918: 01d1            |0006: move v1, v13
08d91a: ee20 0100 1000 |0007: +execute-inline {v0, v1}, [0001] // inline #0001
```

- G1 only uses 4 inlined functions
- invoked with `+execute-inline` opcode
- all are `java.lang.String` operations

num	Method	Signature
0001	charAt	()C
0002	compareTo	(Ljava/lang/String;)I
0003	equals	(Ljava/lang/String;)Z
0004	length	()I

Inlined Functions

execute-inline

```
static void handle_exec_inline(String[] regs, String[] ops,
                        InstructionList il, ConstantPoolGen cpg,
                        LocalVarContext lvg,
                        OpcodeSequence oc, DalvikCodeLine dcl) {
int vtableidx = getvtableidx(ops);
String[] methdata = null;
switch (vtableidx) {
case 1: // String.charAt()
        methdata = new String[] { "java.lang.String", "charAt", "()C" };
        break;
case 2: // String.compareTo()
        methdata = new String[] { "java.lang.String", "compareTo",
        "(Ljava/lang/String;)I" };
        break;
case 3: // String.equals()
        methdata = new String[] { "java.lang.String", "equals",
        "(Ljava/lang/String;)Z" };
        break;
case 4: // String.length()
        methdata = new String[] { "java.lang.String", "length", "()I" };
        break;
default:
        jlog.severe("Unknown inline function");
}
int metref = cpg.addMethodref(methdata[0], methdata[1], methdata[2]);
ClassHandler.genParameterByRegs(il, lvg, regs, methdata, cpg, metref, true);
il.append(new INVOKEVIRTUAL(metref));
```

Vtables

VTable example

```
08a1d8:                  |[08a1d8] java.lang.Object.toString:()Ljava/lang/String;
08a1e8: 2200 3d01        |0000: new-instance v0, Ljava/lang/StringBuilder; //
                          class@013d
08a1ec: 7010 4e0e 0000 |0002: invoke-direct {v0},
                               Ljava/lang/StringBuilder;.<init>:()V // method@0e4e
08a1f2: f810 0300 0200 |0005: +invoke-virtual-quick {v2}, [0003] // vtable #0003
```

- Vtables allow faster dispatch of virtual functions
- Shortcutting superclass resolution
- A class inherits the vtable of its superclasses and appends its own virtual methods

Reconstructing vtables

```
void generate_vtable(String cname, String cnamep) {
        DexClassDetails item = (DexClassDetails) get(cnamep);
        if (item.vtable == null) {
                mergevtable_nocopy(cnamep, item.superclass);
                item = (DexClassDetails) get(cnamep);         }
        DexClassDetails itemp = (DexClassDetails) get(cname);
        MethodCollection vtableneu = new MethodCollection();
        int i = 0;
        for (int ii = 0; ii < item.vtable.size(); ii++,i++) {
                vtableneu.add(ii, item.vtable.get(ii));
        }
        for (int ii = 0; ii < itemp.meths.size(); ii++) {
                DexMethodDetails mi = itemp.meths.get(ii);
                boolean found = false;
                Integer idx = -1;
                int jj = 0 ; for (; jj < item.vtable.size() && !found; jj++) {
                        DexMethodDetails mi2 = item.vtable.get(jj);
                        found = (mi.name.equals(mi2.name) &&
                                (mi.sig.equals(mi2.sig)))
                }
                if (found) {
                        vtableneu.set(idx, mi);
                } else {
                        vtableneu.add(i, mi);
                        i++;
                }
        }
        DexClassDetails newItem = new DexClassDetails(itemp,vtableneu);
        put(cname, newItem);
}
```

Handling calls using precomputed vtables

Using vtables

```
static void handle_invoke_virtual_quick(String[] regs, String[] ops,
        InstructionList il, ConstantPoolGen cpg, LocalVarContext lvg,
    OpcodeSequence oc, DalvikCodeLine dcl) {
        String params = ClassHandler.getparams(regs);
        int vtableidx = getvtableidx(ops);
        String thetype = lvg.getLV(regs[0]).type;
        DexMethodDetails dmd = DalvikToJVM.cc.getVTableEntryForClass(
        thetype,vtableidx);
        String classandmethod = dmd.classname + dmd.name + dmd.sig;
        int metref = cpg.addMethodref(Utils.toJavaName(dmd.classname),
                dmd.name, dmd.sig);
        String[] a = new String[] { dmd.classname, dmd.name, dmd.sig };
        ClassHandler.genParameterByRegs(il, lvg, regs, a, cpg, metref, true);
        il.append(new INVOKEVIRTUAL(metref));
        DalvikCodeLine nextInstr = dcl.getNext();
        if (!nextInstr._opname.startsWith("move-result")
                        && !classandmethod.endsWith(")V")) {
                if (classandmethod.endsWith(")J") ||
                    classandmethod.endsWith(")D")) {
                        il.append(new POP2());
                } else {
                        il.append(new POP());
                }
        }
}
```

Vtables

Field offsets

```
09f904:                |[09f904] java.lang.ClassLoader.definePackage:
                        (Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
                        Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
                        Ljava/lang/String;Ljava/net/URL;)Ljava/lang/Package;
09f914: f4a9 0800      |0000: +iget-object-quick v9, v10, [obj+0008]
09f918: 1d09           |0002: monitor-enter v9
09f91a: f4a1 0800      |0003: +iget-object-quick v1, v10, [obj+0008]
```

- Offsets allow faster access of member fields
- Shortcutting costly name dispatch with integer offset
- A class inherits the fields of its superclasses and appends its own virtual fields

Dealing with optimized field access

Using vtables

```
class FieldCollection extends ArrayList<DexFieldDetails> {
        DexFieldDetails getForOffset(int off) {
                for (int i = 0 ; i < size(); i++) {
                        if (get(i).offset == off) {
                                return get(i);
                        }
                }
                return null;
        }

FieldCollection(String parm) {
        int off = 8;
        for (int i = 0 ; i < size() ; i++) {
                get(i).offset = off;
                int inc =4;
                if (get(i).sig.startsWith("L")) {
                        inc = 8 ;
                }
                if (get(i).sig.startsWith("J")) {
                        inc = 8 ;
                }
                if (get(i).sig.startsWith("D")) {
                        inc = 8 ;
                }
                off += inc;
}
```

Some smaller facts

Hard Facts and Trivia

- 4000 lines of code
- written in JAVA, only external dependency is Apache BCEL
- **undx** name suggested by Dan Bornstein
- command line only
- licensing is **GPL**
- Download at `http://www.illegalaccess.org/undx/`

finally{}

- Thank you for your attention
- Time for Q & A
- or send me a mail

marc.schoenefeld -at- gmx DOT org


HITBSECCONF
2010DUBAI