Manipulating Android Malware to Self-Unpack

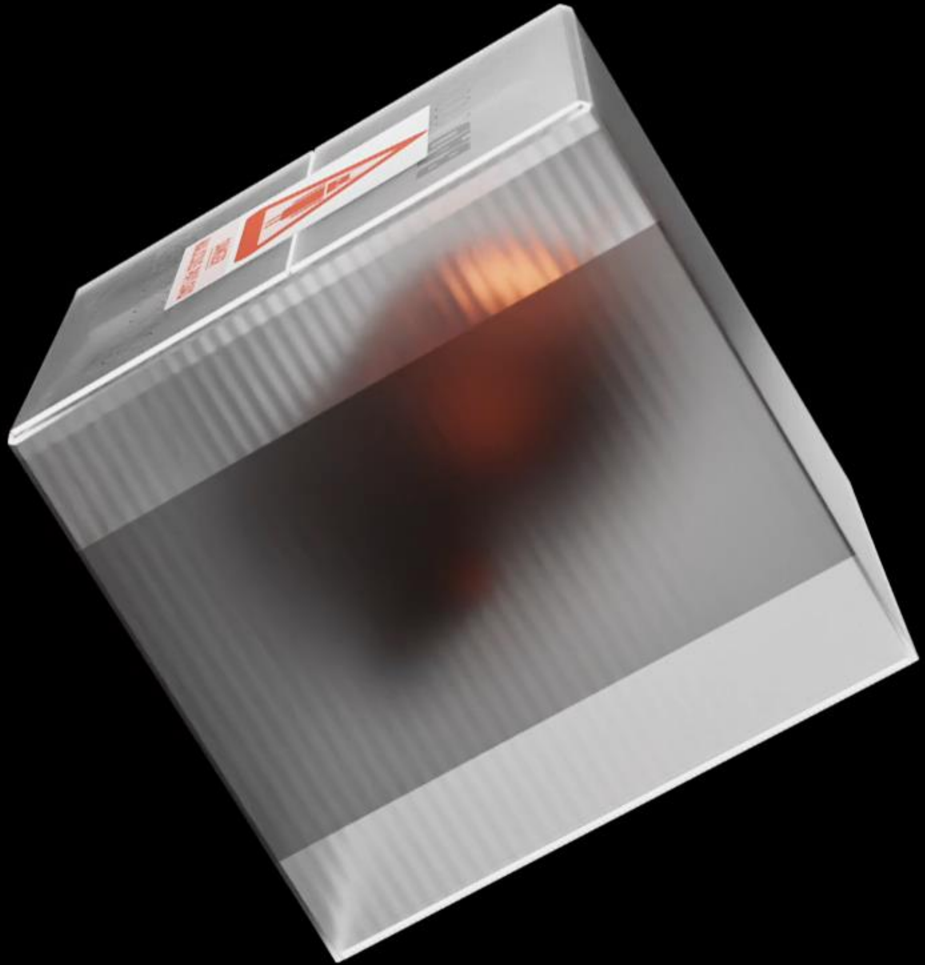
Laurie Kirk

whoami

- Laurie Kirk
- Reverse Engineer
- Specialize in cross-platform malware with a focus on mobile threats
- Run YouTube channel @lauriewired

@lauriewired

SLIDES AND MATERIALS



https://github.com/LaurieWired/RECon2024

33 MILLION

attacks on mobile devices in 2023

SOLUTION

PAYLOAD

DEX FILES PROVIDE UNIQUE OPPORTUNITIES

- Dalvik bytecode is decompiled into Java
- Android builds heavily on common Java APIs
- Custom Android decryptors are written in Java

Opportunity?

Goal:

Defeat Android packers

OUTCOME

- Automate analysis of 1000s of Android samples
- Eliminate reliance on Android emulators
- Remain packer-agnostic

AUTOMATED CUSTOM ANDROID UNPACKERS

Phase 1

Record a Standard Packer Flow

FIND A LARGE SAMPLE SET

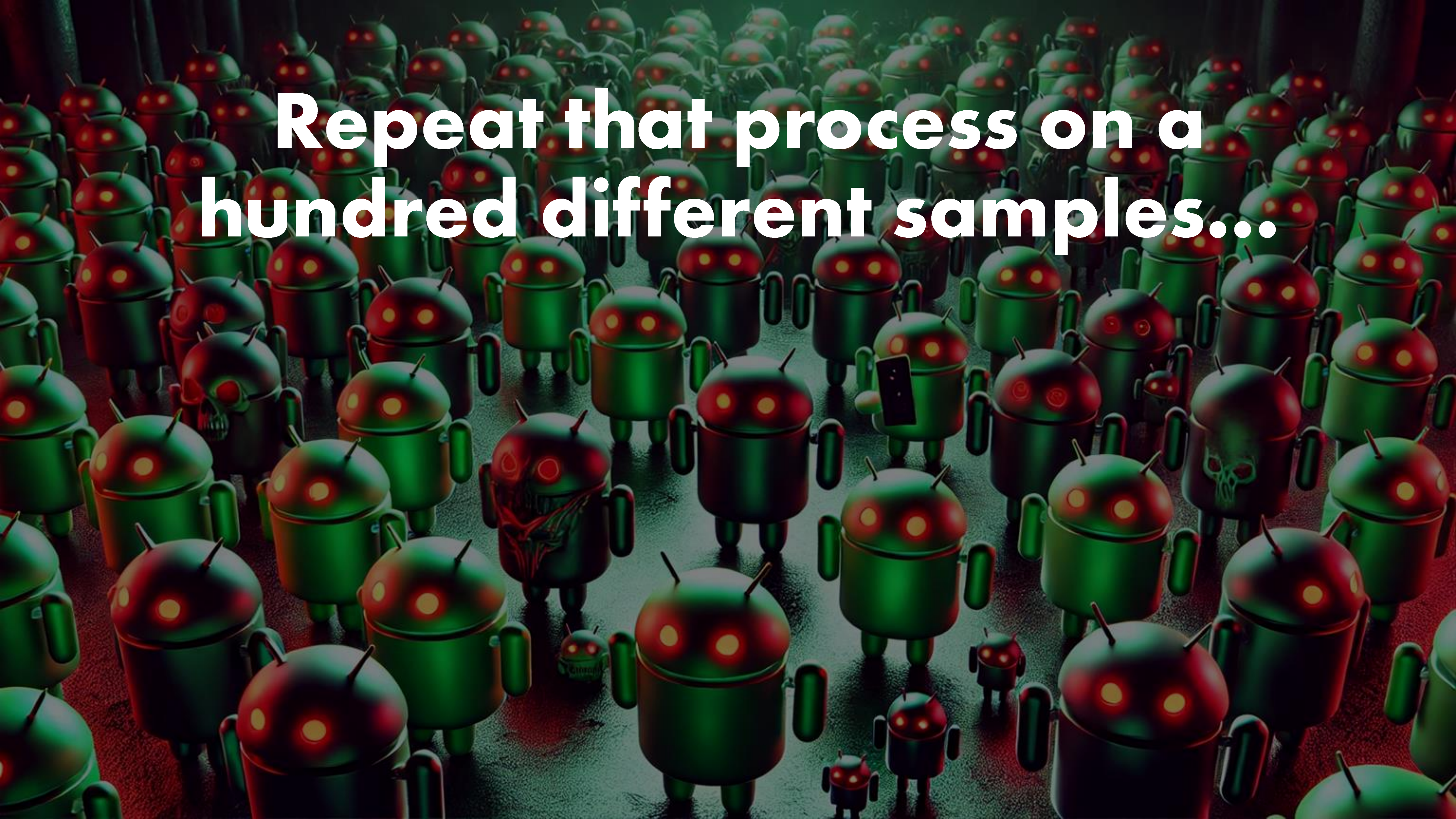
- Need many examples of packers
- Make the unpacking process family-agnostic
- Good candidate: Banking Trojans

ANDROID BANKING TROJANS

- Highly prevalent type of Android malware
- Targets banking / crypto apps to exfiltrate credentials
- Each sample has a unique, custom-generated packing stub

Hands On:

Cerberus Example



Repeat that process on a hundred different samples…

PROCESS SUMMARIZATION

- Manifest classes not defined on disk
- Application subclass contains packing stub
- Dynamic file written to disk / memory
- Stub code calls a ClassLoader
- Dynamic code loaded via Java reflection

Phase 2

Account for Packer Differences

Files can be *dropped* and **loaded** in numerous ways.

REMAINING PACKER AGNOSTIC

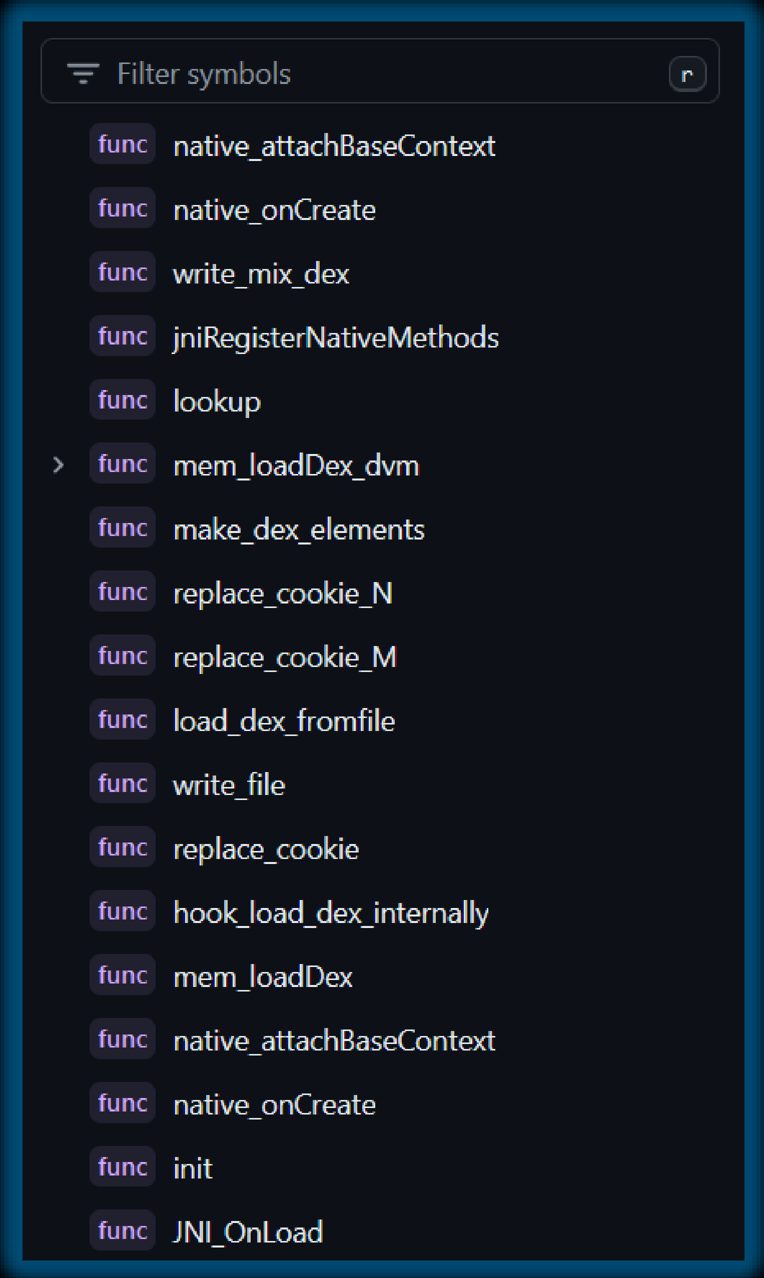
- Account for all standard ClassLoaders
- Handle various techniques for file loading
- Fill anti-debug checks with dummy data

How can I **account** for all of these techniques?

OBSERVE COMMON PACKER SOURCE CODE

- Bangcle Android protector source is on Github
- Older, but methodologies still widely used today
- Multiple *configurable* techniques

DIFFERENT TECHNIQUES IN BANGCLE SOURCE

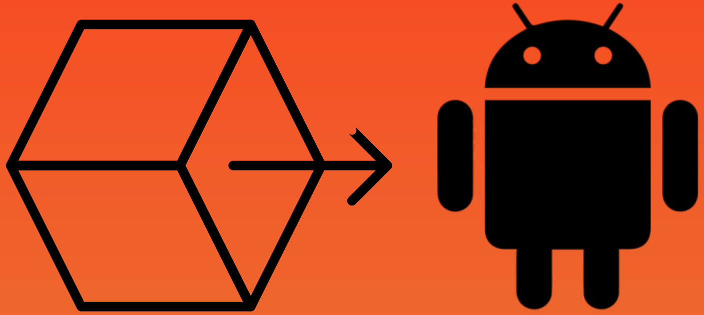


```
func native_attachBaseContext
func native_onCreate
func write_mix_dex
func jniRegisterNativeMethods
func lookup
func mem_loadDex_dvm
func make_dex_elements
func replace_cookie_N
func replace_cookie_M
func load_dex_fromfile
func write_file
func replace_cookie
func hook_load_dex_internally
func mem_loadDex
func native_attachBaseContext
func native_onCreate
func init
func JNI_OnLoad
```

RELEVANT API CALLS

- ClassLoaders
- Dexfile
- OpenMemory
- ZipEntry

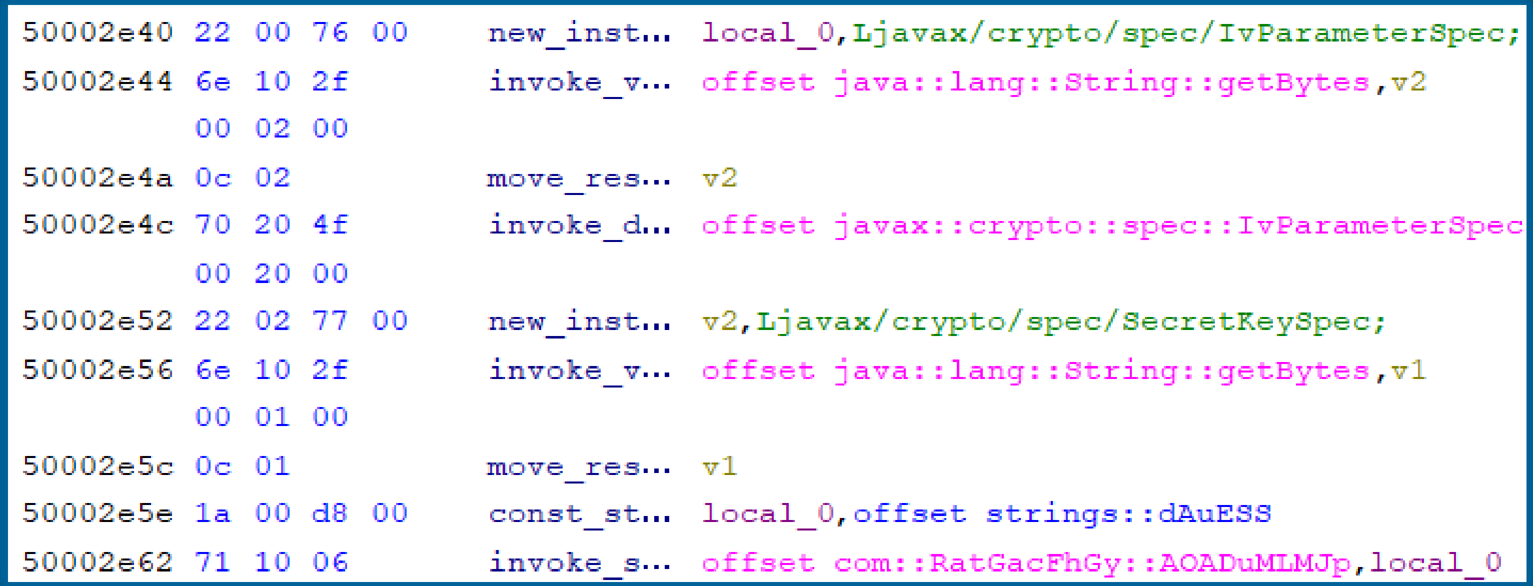
Phase 3



Automate the Unpacking Process

Idea:
Patch the APK

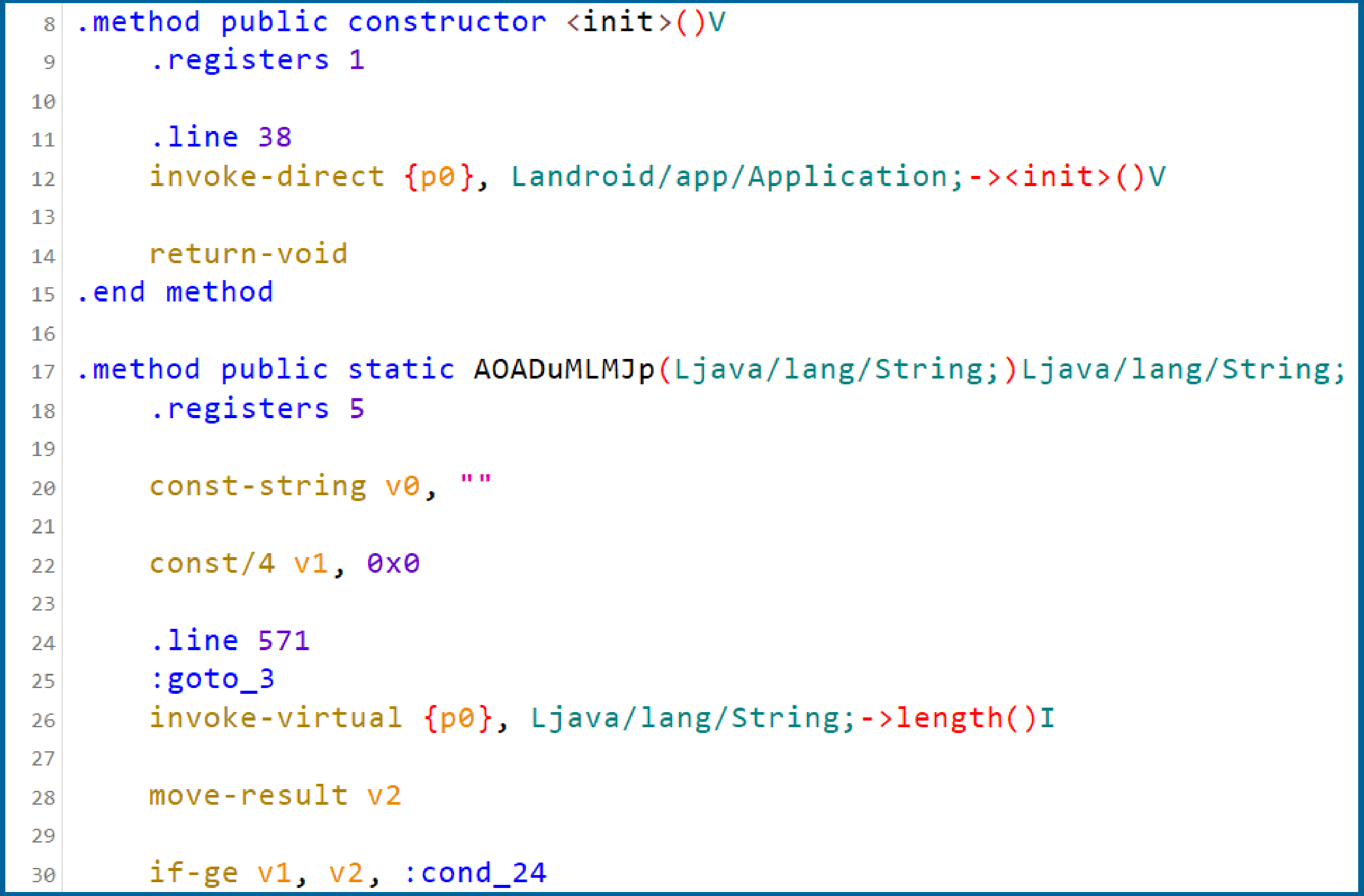
OPTION 1:
Patching Bytes in classes.dex



```
50002e40 22 00 76 00     new_inst… local_0,Ljavax/crypto/spec/IvParameterSpec;
50002e44 6e 10 2f        invoke_v… offset java::lang::String::getBytes,v2
         00 02 00
50002e4a 0c 02           move_res… v2
50002e4c 70 20 4f        invoke_d… offset javax::crypto::spec::IvParameterSpec
         00 20 00
50002e52 22 02 77 00     new_inst… v2,Ljavax/crypto/spec/SecretKeySpec;
50002e56 6e 10 2f        invoke_v… offset java::lang::String::getBytes,v1
         00 01 00
50002e5c 0c 01           move_res… v1
50002e5e 1a 00 d8 00     const_st… local_0,offset strings::dAuESS
50002e62 71 10 06        invoke_s… offset com::RatGacFhGy::AOADuMLMJp,local_0
```

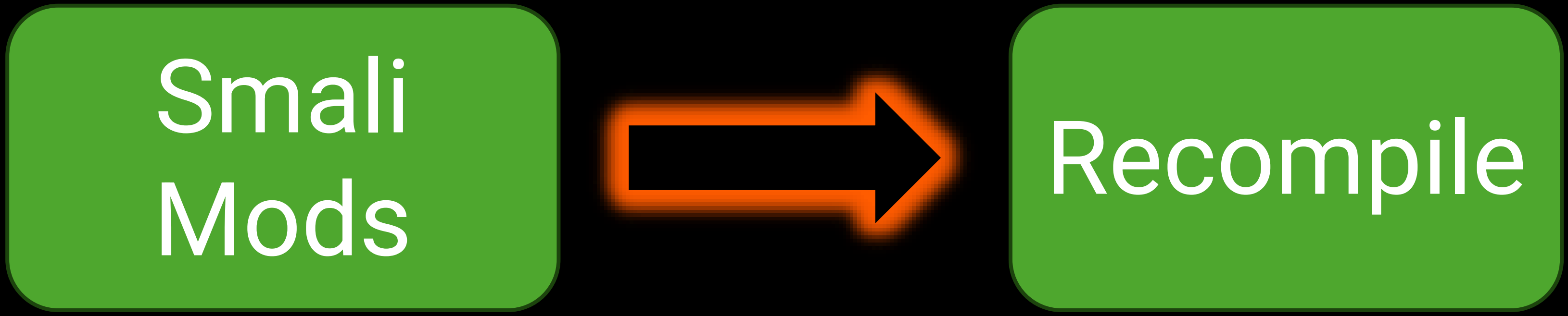
OPTION 2:

Editing Smali



```
.method public constructor <init>()V
    .registers 1

    .line 38
    invoke-direct {p0}, Landroid/app/Application;-><init>()V

    return-void
.end method

.method public static AOADuMLMJp(Ljava/lang/String;)Ljava/lang/String;
    .registers 5

    const-string v0, ""

    const/4 v1, 0x0

    .line 571
    :goto_3
    invoke-virtual {p0}, Ljava/lang/String;->length()I

    move-result v2

    if-ge v1, v2, :cond_24
```

APK MODS



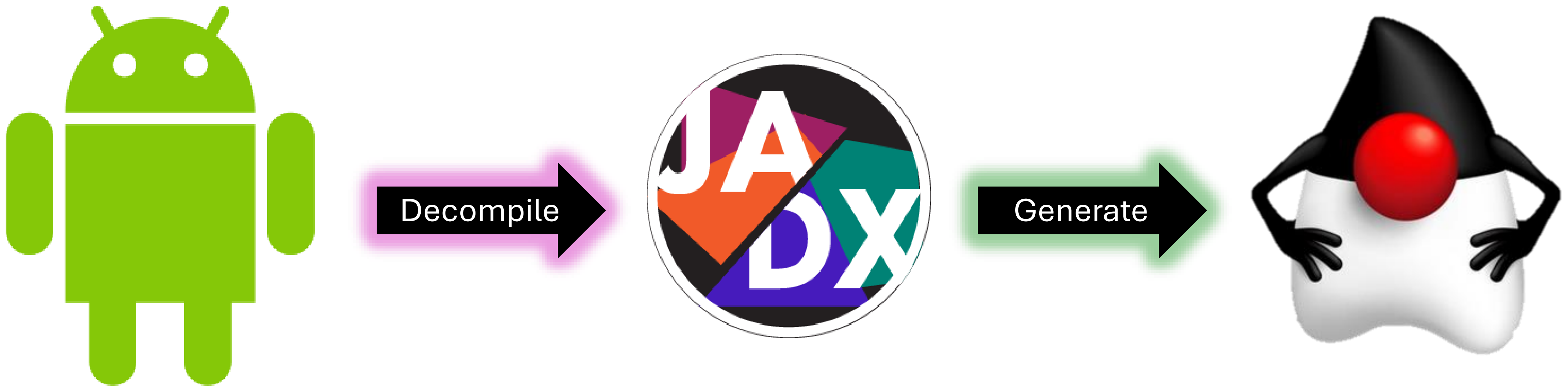
DRAWBACKS

- Smali editing is tedious
- Apps must be re-signed prior to dynamic analysis
- Both still require an Android emulator

This is **NOT** the way

Idea:

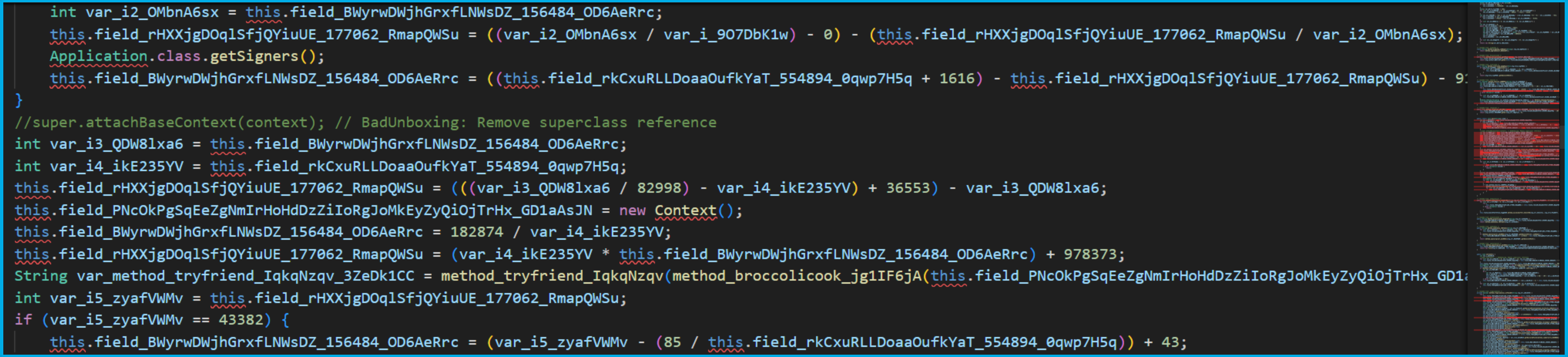
Generate Unpacker Code from Decompiled Code


Decompile
JADX
Generate

LOCATING THE RELEVANT CODE

Decompile	Decompile the stub app
Find Entry	Grab the Application subclass
References	Find references to other classes
Unpacker	Generate the custom unpacker in Java

THAT'S A LOT OF ERRORS...



```
    int var_i2_OMbnA6sx = this.field_BWyrwDWjhGrxfLNWsDZ_156484_OD6AeRrc;
    this.field_rHXXjgDOqlSfjQYiuUE_177062_RmapQWSu = ((var_i2_OMbnA6sx / var_i_9O7DbK1w) - 0) - (this.field_rHXXjgDOqlSfjQYiuUE_177062_RmapQWSu / var_i2_OMbnA6sx);
    Application.class.getSigners();
    this.field_BWyrwDWjhGrxfLNWsDZ_156484_OD6AeRrc = ((this.field_rkCxuRLLDoaaOufkYaT_554894_0qwp7H5q + 1616) - this.field_rHXXjgDOqlSfjQYiuUE_177062_RmapQWSu) - 9
}
//super.attachBaseContext(context); // BadUnboxing: Remove superclass reference
int var_i3_QDW8lxa6 = this.field_BWyrwDWjhGrxfLNWsDZ_156484_OD6AeRrc;
int var_i4_ikE235YV = this.field_rkCxuRLLDoaaOufkYaT_554894_0qwp7H5q;
this.field_rHXXjgDOqlSfjQYiuUE_177062_RmapQWSu = (((var_i3_QDW8lxa6 / 82998) - var_i4_ikE235YV) + 36553) - var_i3_QDW8lxa6;
this.field_PNcOkPgSqEeZgNmIrHoHdDzZiIoRgJoMkEyZyQiOjTrHx_GD1aAsJN = new Context();
this.field_BWyrwDWjhGrxfLNWsDZ_156484_OD6AeRrc = 182874 / var_i4_ikE235YV;
this.field_rHXXjgDOqlSfjQYiuUE_177062_RmapQWSu = (var_i4_ikE235YV * this.field_BWyrwDWjhGrxfLNWsDZ_156484_OD6AeRrc) + 978373;
String var_method_tryfriend_IqkqNzqv_3ZeDk1CC = method_tryfriend_IqkqNzqv(method_broccolicook_jg1IF6jA(this.field_PNcOkPgSqEeZgNmIrHoHdDzZiIoRgJoMkEyZyQiOjTrHx_GD1a
int var_i5_zyafVWMv = this.field_rHXXjgDOqlSfjQYiuUE_177062_RmapQWSu;
if (var_i5_zyafVWMv == 43382) {
    this.field_BWyrwDWjhGrxfLNWsDZ_156484_OD6AeRrc = (var_i5_zyafVWMv - (85 / this.field_rkCxuRLLDoaaOufkYaT_554894_0qwp7H5q)) + 43;
```

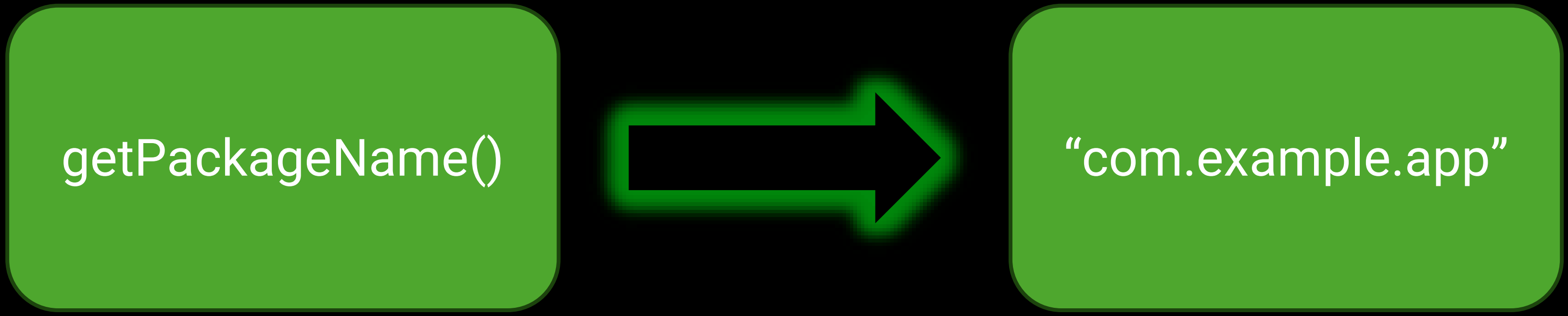
Problem!

Java doesn’t understand Android API calls.

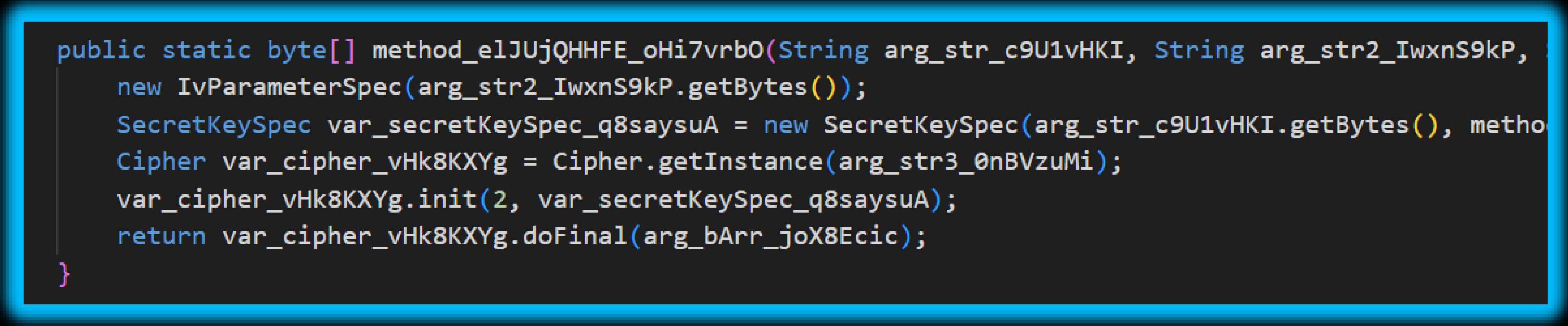
PLAN:

Replace Android APIs with equivalent Java

HARDCODING COMMON ANDROID STRINGS



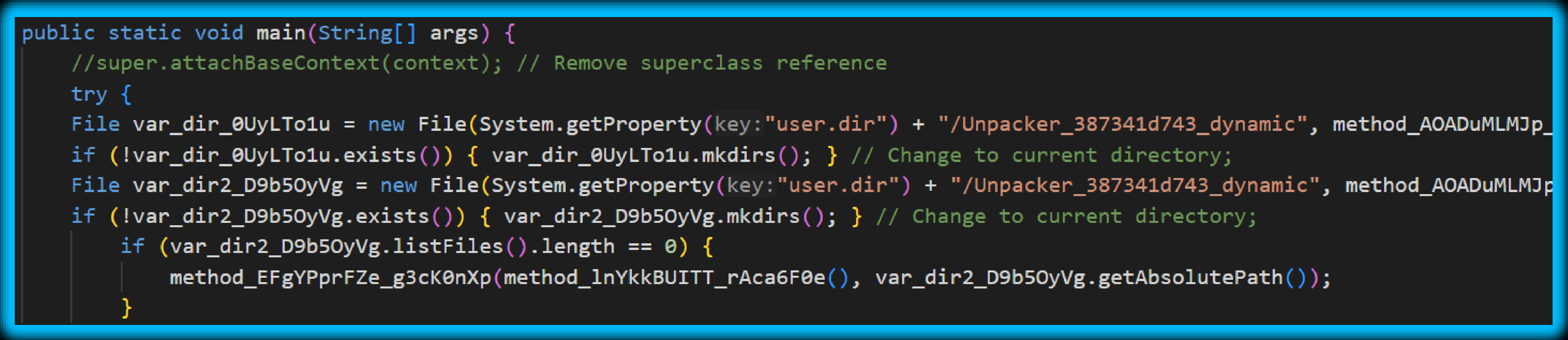
IMPORTANT:
LEAVE IN GENERIC DECRYPTION CODE



```
public static byte[] method_elJUjQHHFE_oHi7vrbO(String arg_str_c9U1vHKI, String arg_str2_IwxnS9kP,
    new IvParameterSpec(arg_str2_IwxnS9kP.getBytes());
    SecretKeySpec var_secretKeySpec_q8saysuA = new SecretKeySpec(arg_str_c9U1vHKI.getBytes(), metho
    Cipher var_cipher_vHk8KXYg = Cipher.getInstance(arg_str3_0nBVzuMi);
    var_cipher_vHk8KXYg.init(2, var_secretKeySpec_q8saysuA);
    return var_cipher_vHk8KXYg.doFinal(arg_bArr_joX8Ecic);
}
```

EXAMPLE:

REPLACING ANDROID FILE CALLS WITH CURRENT DIRECTORY



```
public static void main(String[] args) {
    //super.attachBaseContext(context); // Remove superclass reference
    try {
    File var_dir_0UyLTo1u = new File(System.getProperty(key:"user.dir") + "/Unpacker_387341d743_dynamic", method_AOADuMLMJp_
    if (!var_dir_0UyLTo1u.exists()) { var_dir_0UyLTo1u.mkdirs(); } // Change to current directory;
    File var_dir2_D9b5OyVg = new File(System.getProperty(key:"user.dir") + "/Unpacker_387341d743_dynamic", method_AOADuMLMJp
    if (!var_dir2_D9b5OyVg.exists()) { var_dir2_D9b5OyVg.mkdirs(); } // Change to current directory;
        if (var_dir2_D9b5OyVg.listFiles().length == 0) {
            method_EFgYPprFZe_g3cK0nXp(method_lnYkkBUITT_rAca6F0e(), var_dir2_D9b5OyVg.getAbsolutePath());
        }
```

Reflection is not specific to Android.

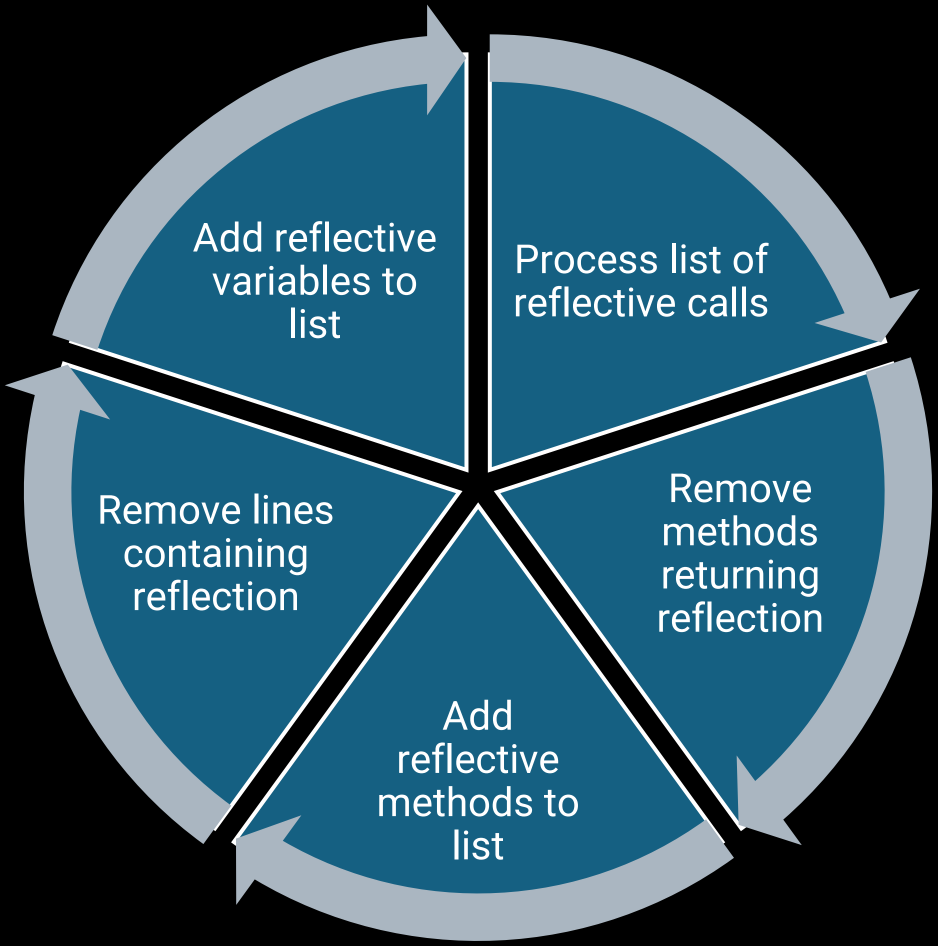
REFLECTION

- Feature in both plain Java and Android
- Allows programs to introspect themselves
- Enables dynamic code loading

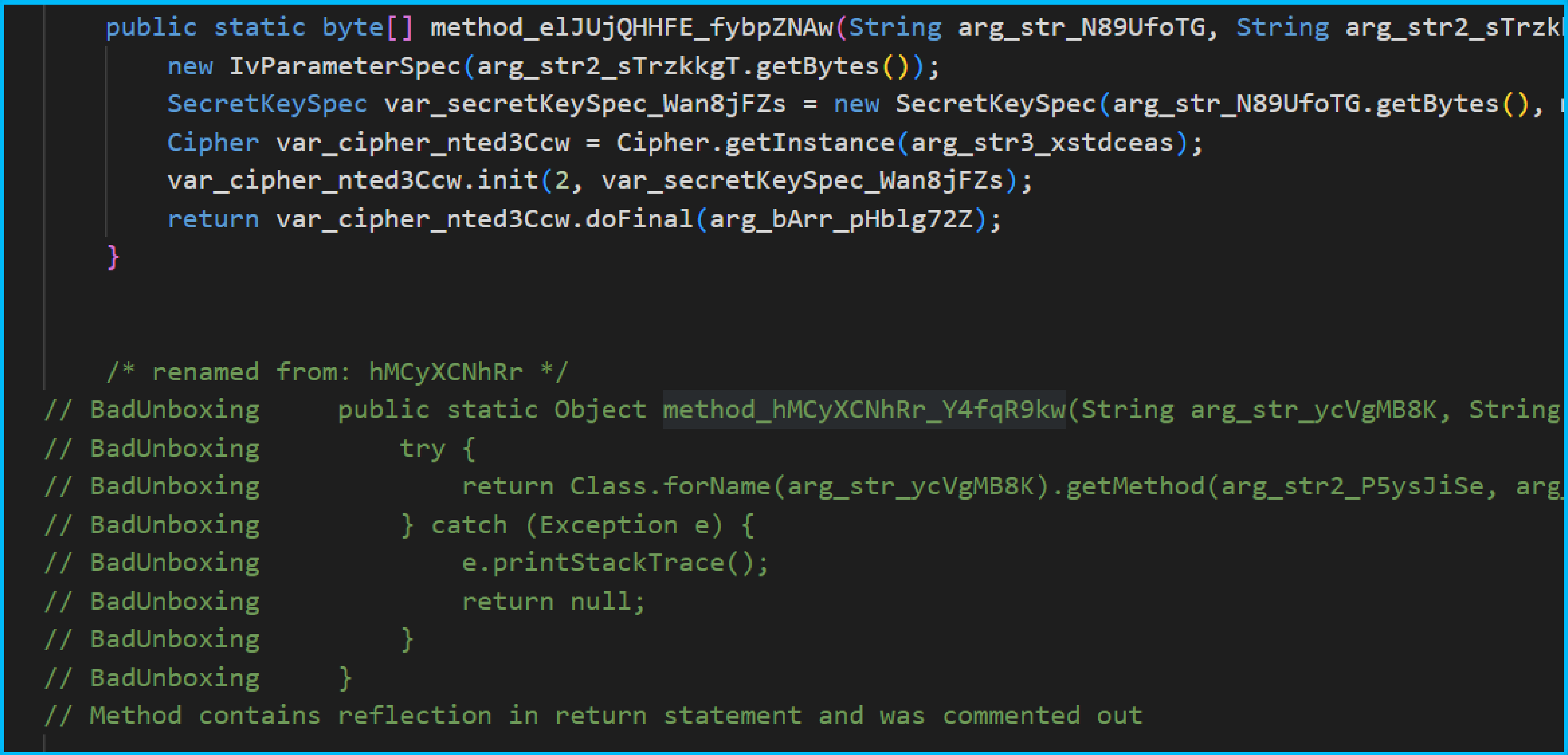
I need to remove **reflective** calls and calls to **reflective** calls.

Let’s get recursive.

REMOVING REFLECTIVE JAVA CALLS

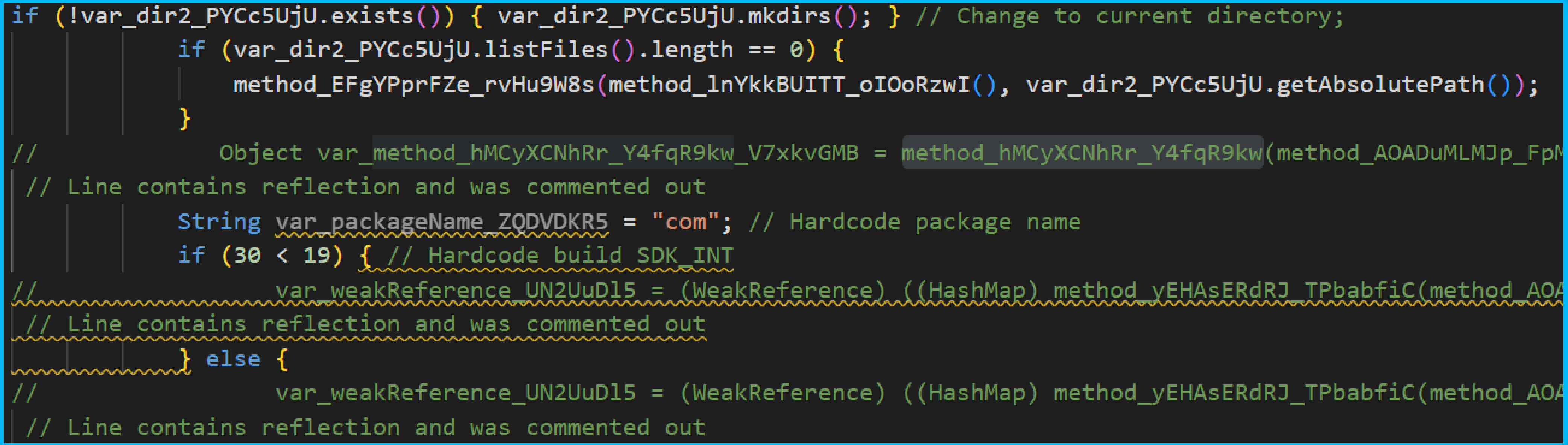

Process list of reflective calls
Remove methods returning reflection
Add reflective methods to list
Remove lines containing reflection
Add reflective variables to list

REMOVE REFLECTIVE METHOD



```
    public static byte[] method_elJUjQHHFE_fybpZNAw(String arg_str_N89UfoTG, String arg_str2_sTrzk
        new IvParameterSpec(arg_str2_sTrzkkgT.getBytes());
        SecretKeySpec var_secretKeySpec_Wan8jFZs = new SecretKeySpec(arg_str_N89UfoTG.getBytes(), 
        Cipher var_cipher_nted3Ccw = Cipher.getInstance(arg_str3_xstdceas);
        var_cipher_nted3Ccw.init(2, var_secretKeySpec_Wan8jFZs);
        return var_cipher_nted3Ccw.doFinal(arg_bArr_pHblg72Z);
    }


    /* renamed from: hMCyXCNhRr */
// BadUnboxing     public static Object method_hMCyXCNhRr_Y4fqR9kw(String arg_str_ycVgMB8K, String
// BadUnboxing         try {
// BadUnboxing             return Class.forName(arg_str_ycVgMB8K).getMethod(arg_str2_P5ysJiSe, arg
// BadUnboxing         } catch (Exception e) {
// BadUnboxing             e.printStackTrace();
// BadUnboxing             return null;
// BadUnboxing         }
// BadUnboxing     }
// Method contains reflection in return statement and was commented out
```

ADD METHOD NAME TO REFLECTION KEYWORD LIST



```
if (!var_dir2_PYCc5UjU.exists()) { var_dir2_PYCc5UjU.mkdirs(); } // Change to current directory;
      if (var_dir2_PYCc5UjU.listFiles().length == 0) {
        method_EFgYPprFZe_rvHu9W8s(method_lnYkkBUITT_oIOoRzwI(), var_dir2_PYCc5UjU.getAbsolutePath());
      }
//      Object var_method_hMCyXCNhRr_Y4fqR9kw_V7xkvGMB = method_hMCyXCNhRr_Y4fqR9kw(method_AOADuMLMJp_FpM
 // Line contains reflection and was commented out
      String var_packageName_ZQDVDKR5 = "com"; // Hardcode package name
      if (30 < 19) { // Hardcode build SDK_INT
//        var_weakReference_UN2UuDl5 = (WeakReference) ((HashMap) method_yEHAsERdRJ_TPbabfiC(method_AOA
 // Line contains reflection and was commented out
      } else {
//        var_weakReference_UN2UuDl5 = (WeakReference) ((HashMap) method_yEHAsERdRJ_TPbabfiC(method_AOA
 // Line contains reflection and was commented out
```

PROCESS SUMMARIZATION SO FAR

- App subclass becomes Java app
- Decompile dependencies from APK
- Remove Android imports
- Replace Android APIs with Java
- Remove reflection calls

Phase 4

Perform these processes by hand

~~Phase 4~~

~~Perform these processes by hand~~

Phase 4

Write a tool to perform this process

Introducing

BADUnboxing

BADUNBOXING FEATURES

1. Detect packing
2. Extract and decompile relevant code
3. Replace Android API calls
4. Eliminate reflective calls
5. Generate custom Java unpacker

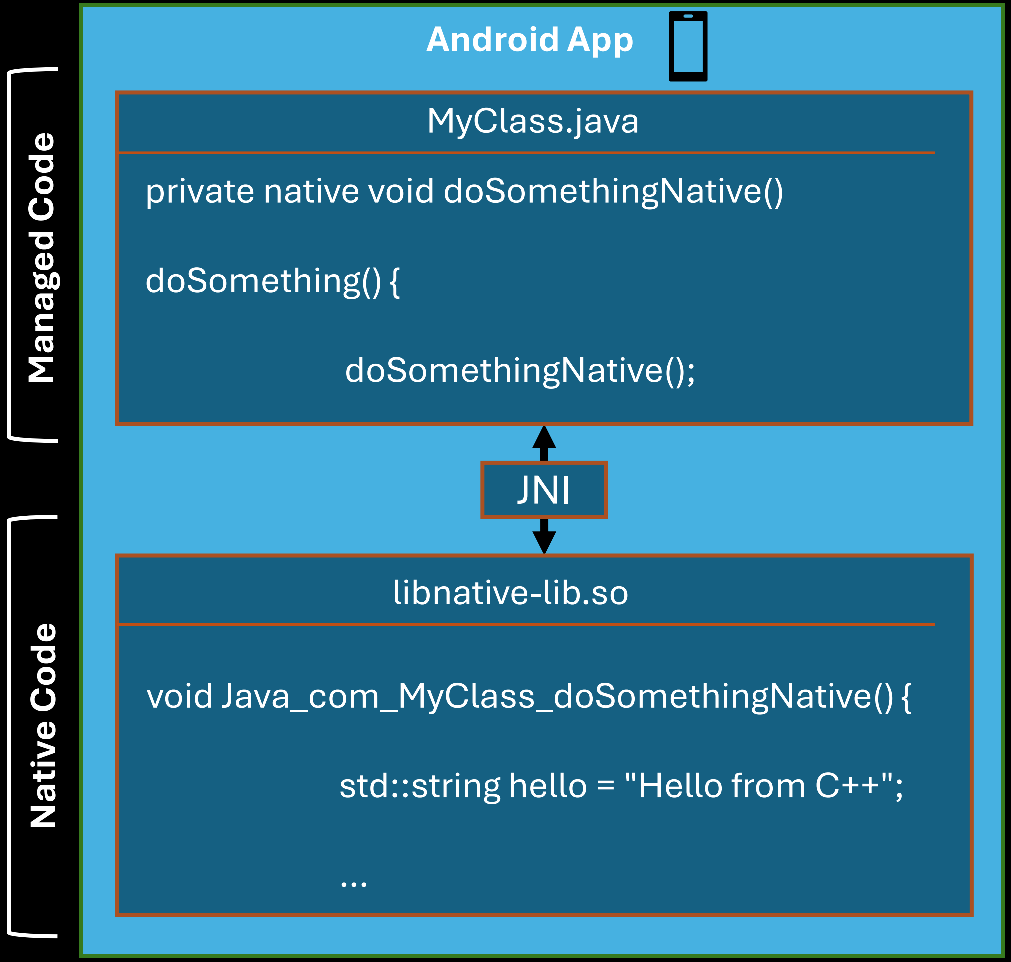
DEMO:

Auto Unpacking Android Malware with BadUnboxing

Shift

Towards Native Packing

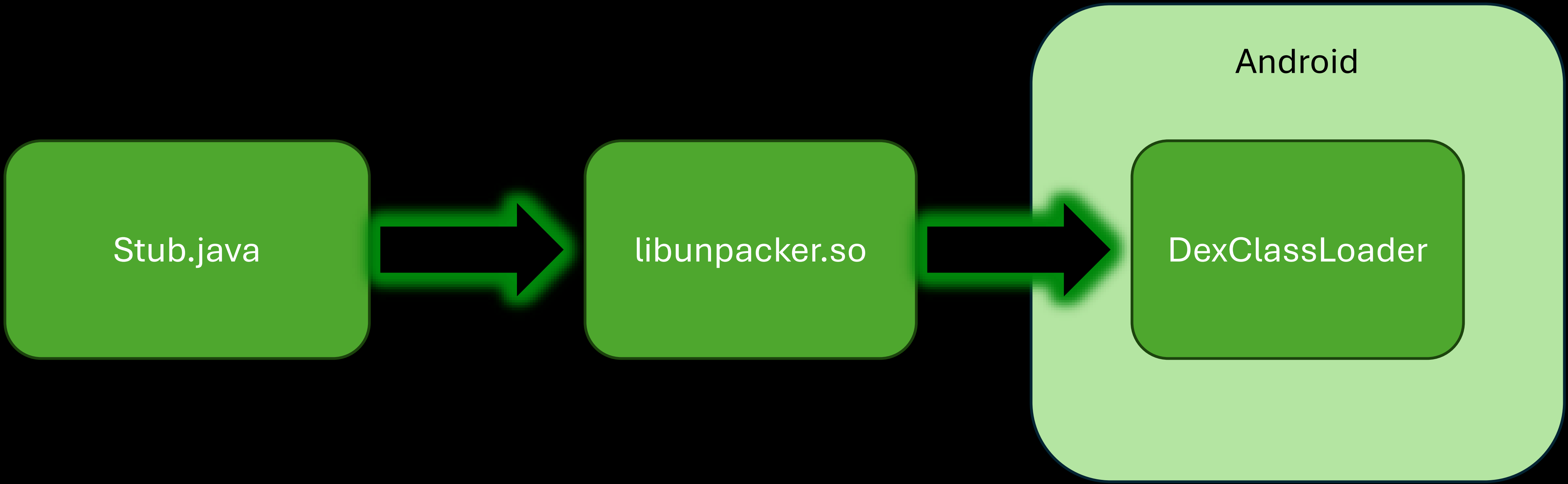
NATIVE PACKING



```
private native void doSomethingNative()

doSomething() {

        doSomethingNative();
```

```
void Java_com_MyClass_doSomethingNative() {

        std::string hello = "Hello from C++";

        ...
```

NATIVE PACKING



The JNI is also a standard Java construct.

Native code without Android APIs can be called **directly.**

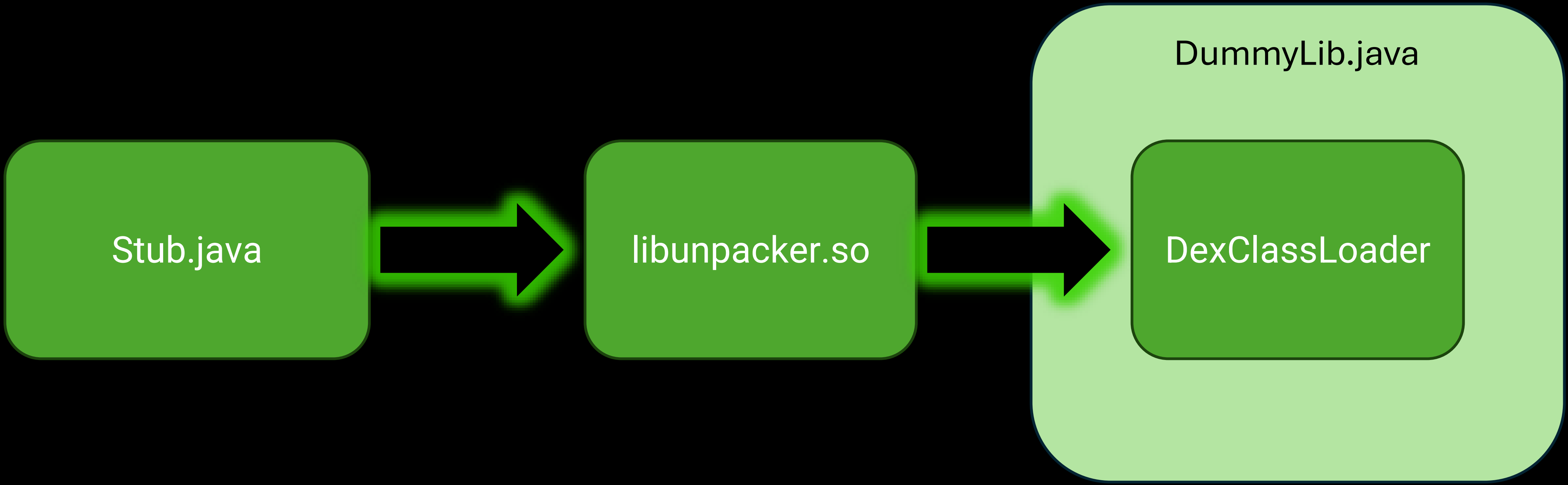
Problem!

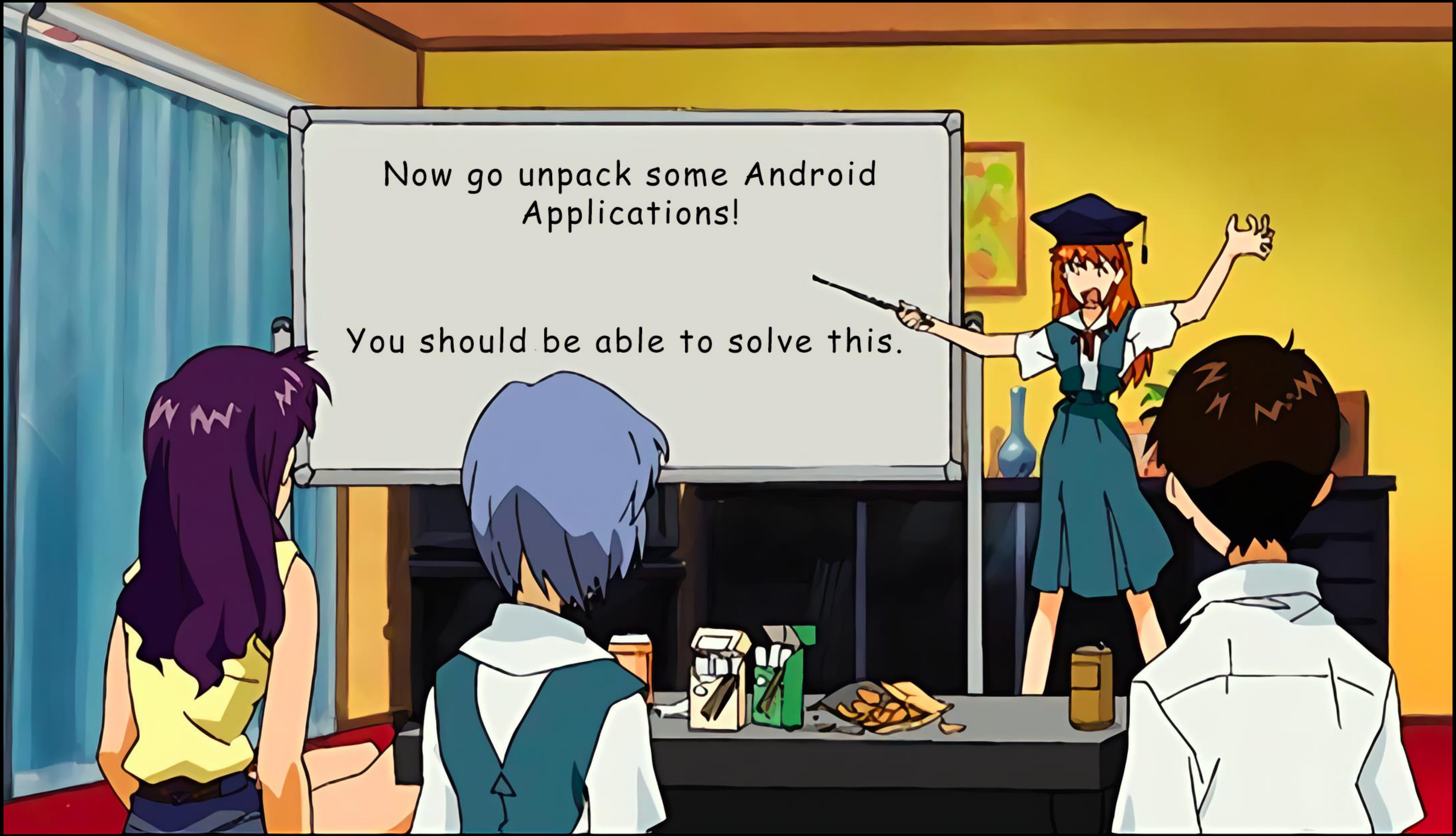
What about native code with Android API calls?

PLAN:

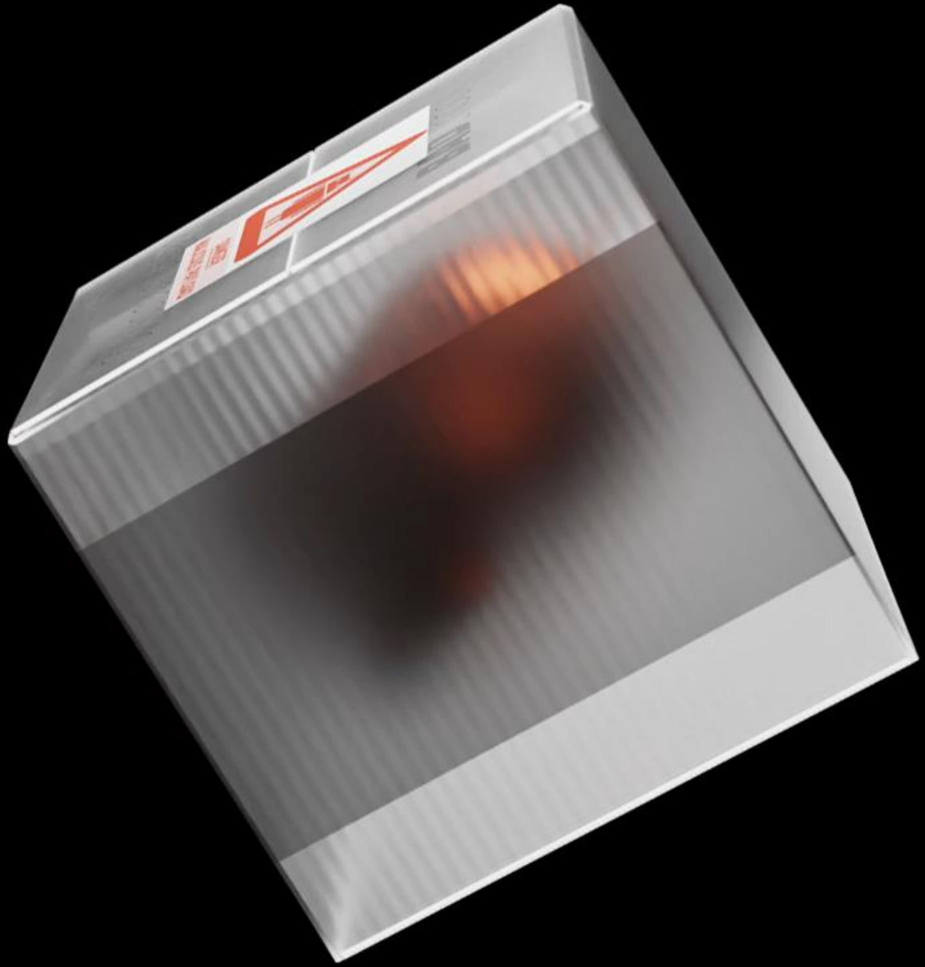
Implement Dummy Android APIs in Java

NATIVE PACKING




Now go unpack some Android Applications!
You should be able to solve this.

THANK YOU!



https://github.com/LaurieWired/BadUnboxing