Runtime Riddles

Abusing Manipulation Points in the Android Source

whoami

- Laurie Kirk
- Reverse Engineer at Microsoft
- Specialize in cross-platform malware with a focus on mobile malware
- Run YouTube channel @lauriewired
- Representing myself as an individual security researcher today (not representing Microsoft)

@lauriewired

Analysis Materials

- LaurieWired DEF CON Github Repo
  - https://github.com/LaurieWired/RuntimeRiddles_DEFCON

Imagine you’re a seasoned security analyst

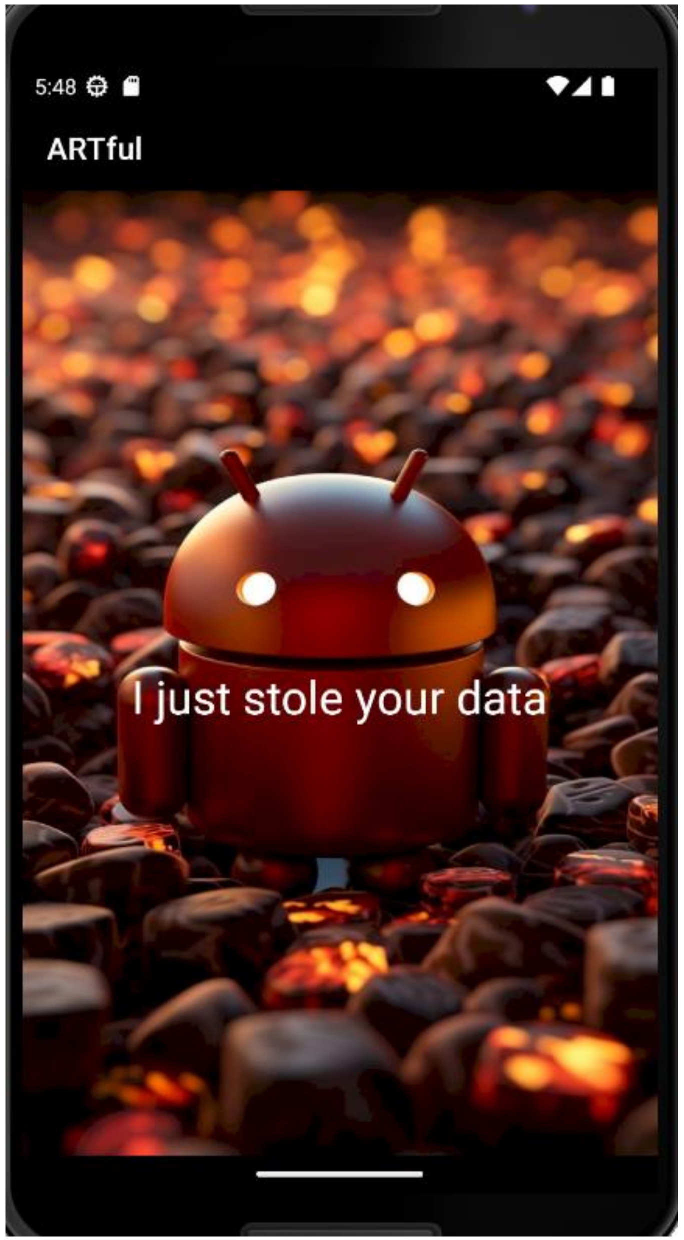
Analyzing a seemingly benign app

Every referenced method

Checks out ✓

Is this code safe?

```java
Log.e("ARTful", "Starting app");
Button button = new Button(this);
button.setText("Click Me");
Log.e("ARTful", "Created new button");
```



I just stole your data

Agenda

- Manipulate the Android13 runtime
- Replace Android APIs in apps with hidden “malicious” code
- Provide new open-source tool to the community
- Defeat reverse engineers

Dynamic Obfuscation Goals

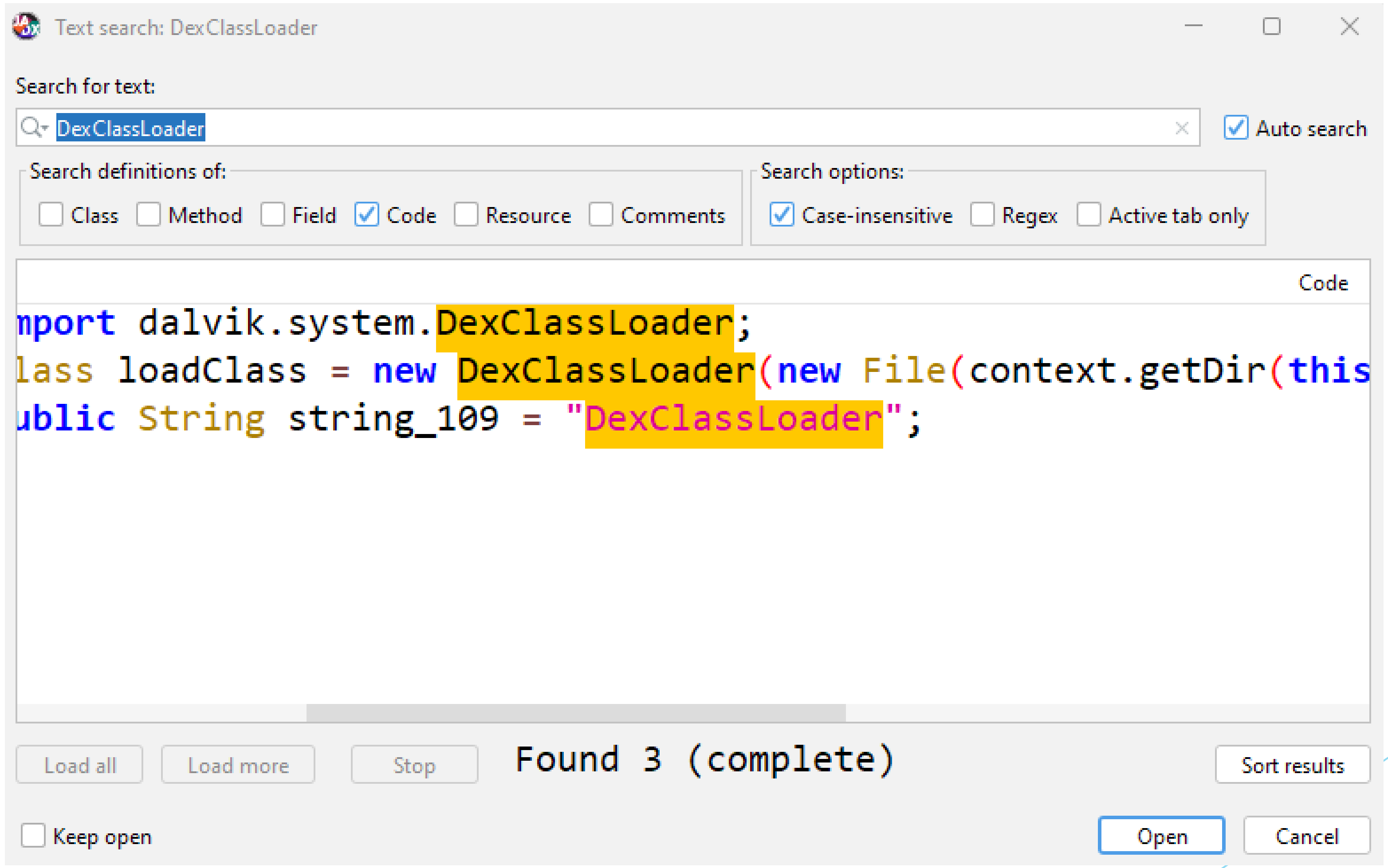
- Load dynamic code
- Prevent static analysis

Dynamic Obfuscation Options

- DexClassLoader
- PathClassLoader
- ClassLoader.loadClass

These successfully alter control flow

Searchable in plaintext



Search for text: DexClassLoader

[✓ Auto search]

Search definitions of: Class, Method, Field, [✓ Code], Resource, Comments

Search options: [✓ Case-insensitive], Regex, Active tab only

```
mport dalvik.system.DexClassLoader;
lass loadClass = new DexClassLoader(new File(context.getDir(this
ublic String string_109 = "DexClassLoader";
```

Found 3 (complete)

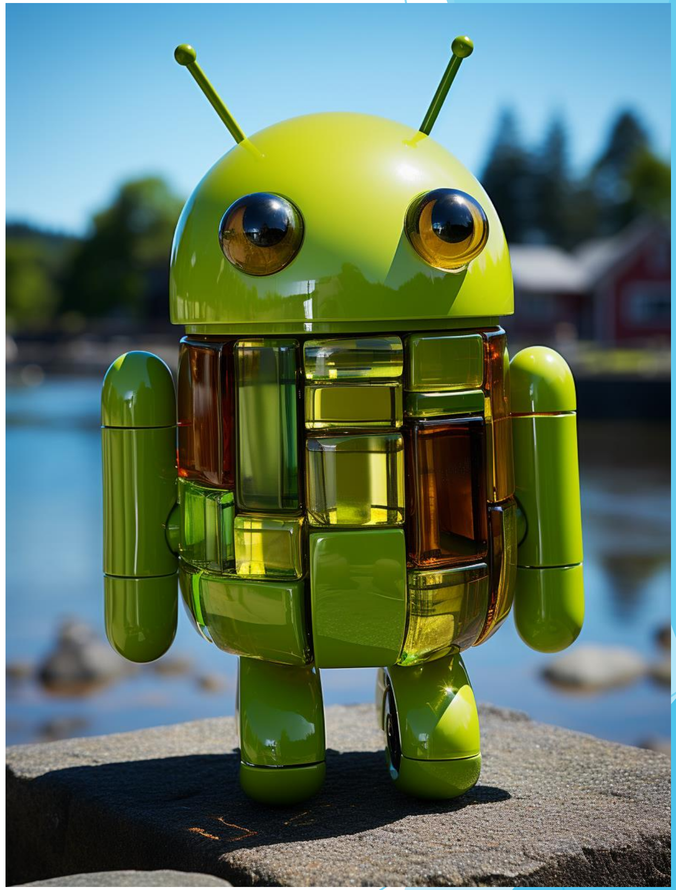
Automatically hooked and analyzed

Can we bypass standard API calls?

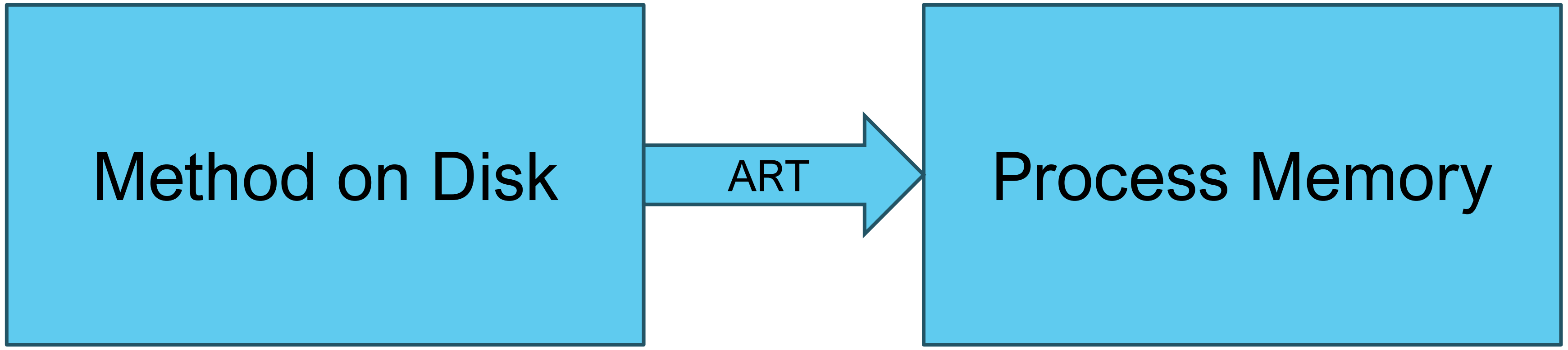
Manipulating ART

ART Architecture

- Android Runtime
- Transforms executables into runtime objects
- Loads and executes Android apps



From Disk to Memory



Idea:
Manipulate Methods in Memory

This should produce:

No plaintext Android API calls ✓

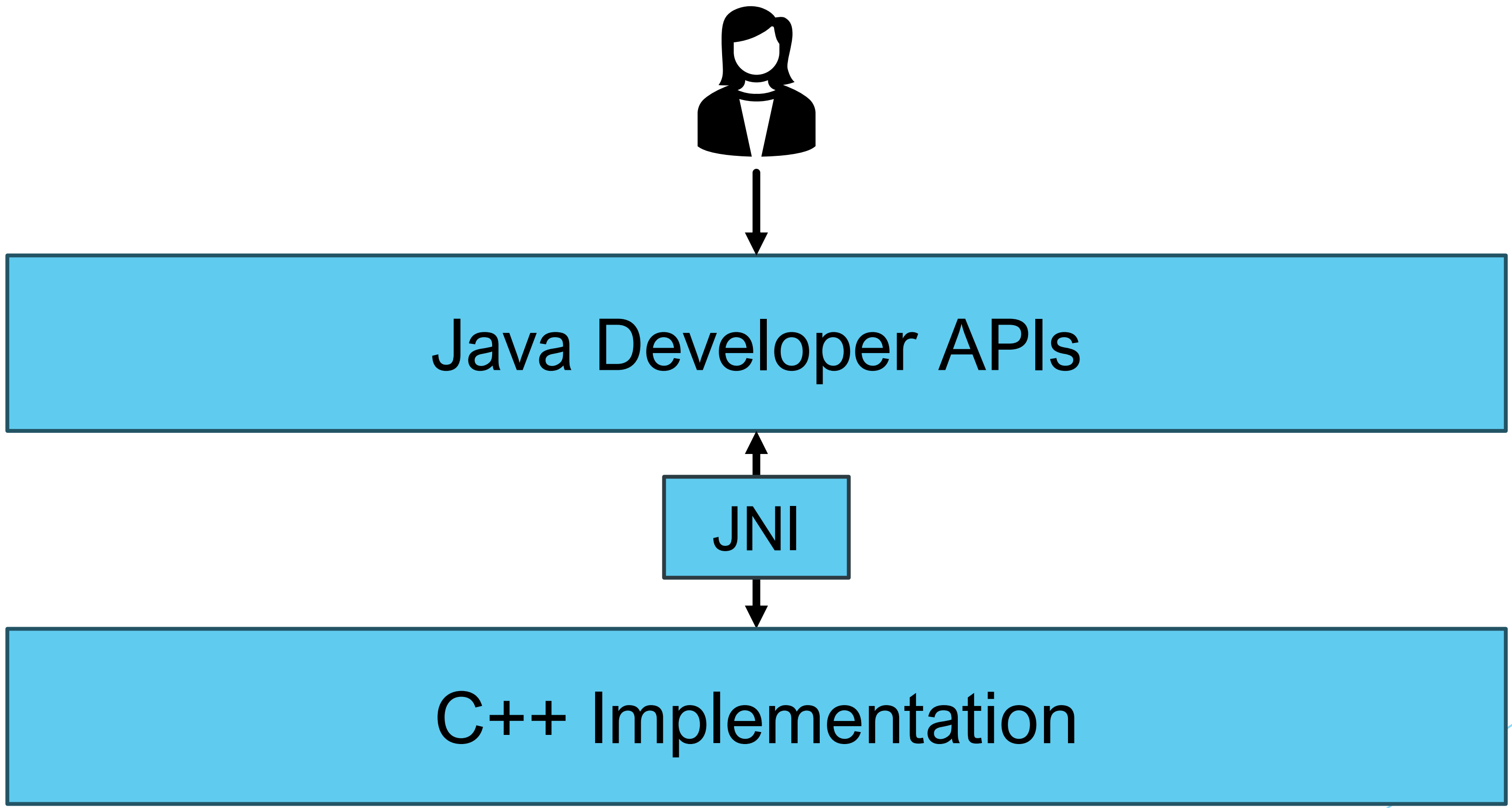
No standard methods to hook ✓

Android has many abstraction layers.

Android’s Abstractions

- Android APIs are exposed in Java
- Java wraps C++ implementation

Developer View



We must modify C++ in the Android Framework.

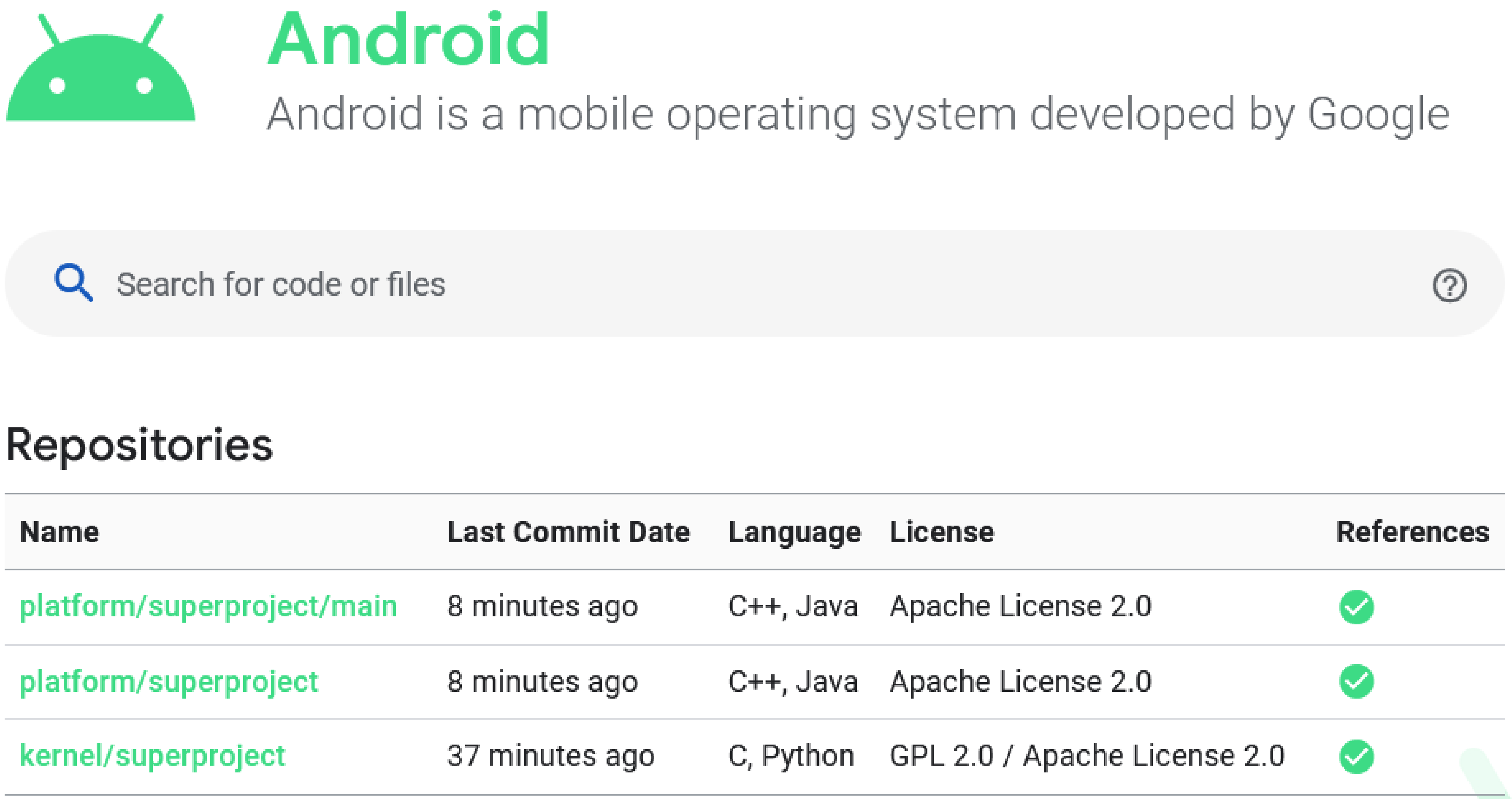
ART Modification Methodology

Locate Java Target

Intercept via JNI

Overwrite native data

Finding Android Framework Targets



Android

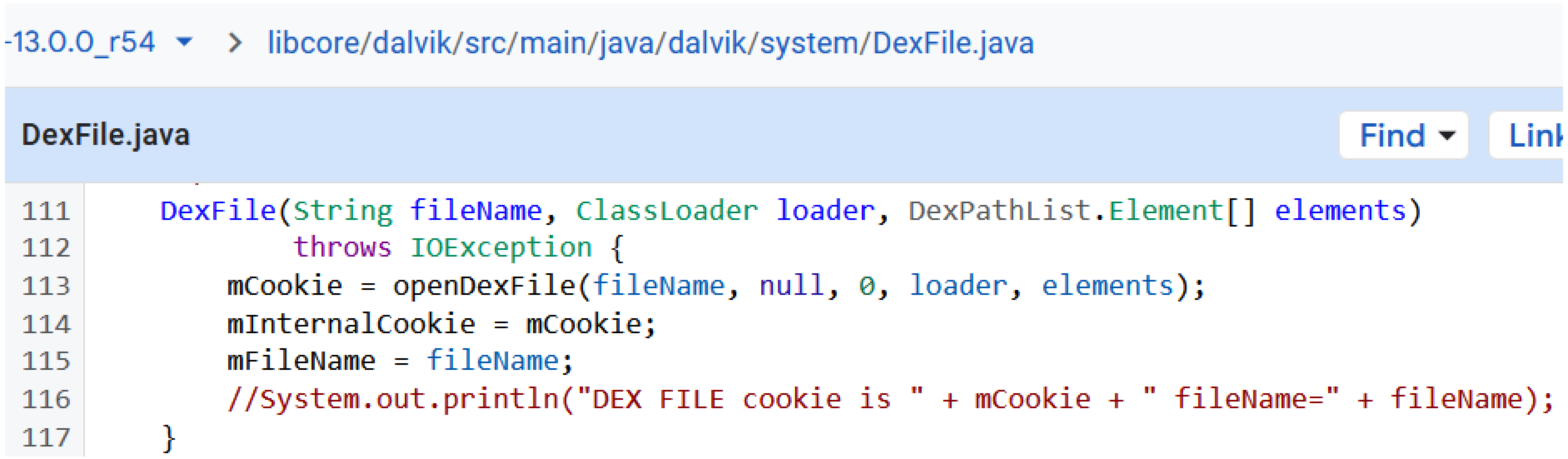
Android is a mobile operating system developed by Google

Search for code or files

Repositories

Name	Last Commit Date	Language	License	References
platform/superproject/main	8 minutes ago	C++, Java	Apache License 2.0	✓
platform/superproject	8 minutes ago	C++, Java	Apache License 2.0	✓
kernel/superproject	37 minutes ago	C, Python	GPL 2.0 / Apache License 2.0	✓

Target: Swap Entire DEX file



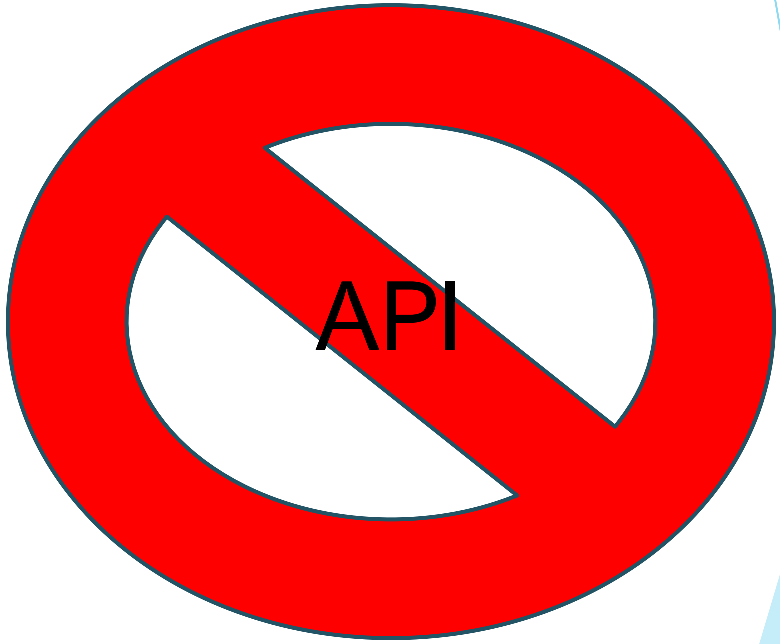
-13.0.0_r54 > libcore/dalvik/src/main/java/dalvik/system/DexFile.java

DexFile.java

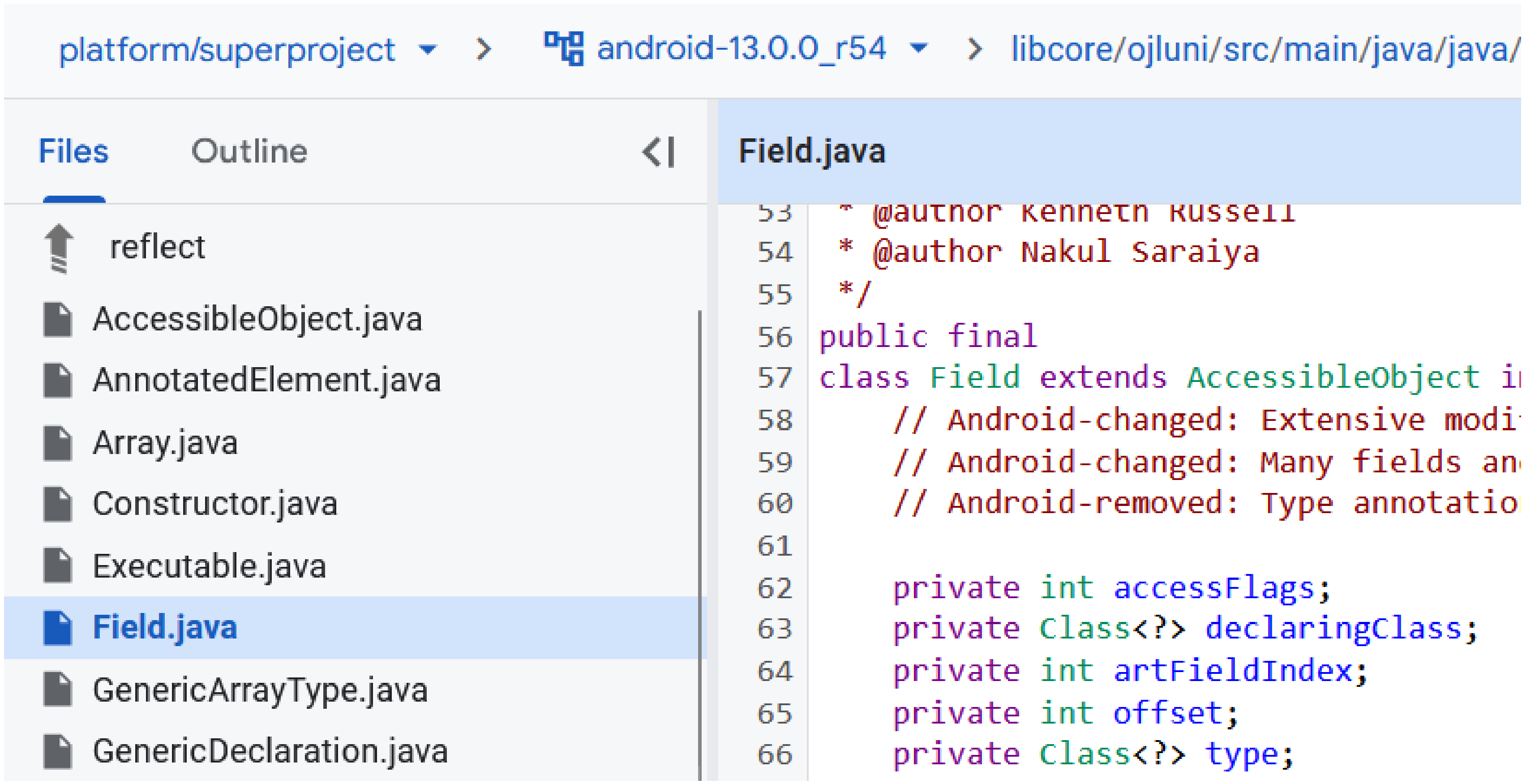
```
    DexFile(String fileName, ClassLoader loader, DexPathList.Element[] elements)
            throws IOException {
        mCookie = openDexFile(fileName, null, 0, loader, elements);
        mInternalCookie = mCookie;
        mFileName = fileName;
        //System.out.println("DEX FILE cookie is " + mCookie + " fileName=" + fileName);
    }
```

Problems

- Main DEX file is already loaded
- Additional files require ClassLoader call



Modify Individual Members


platform/superproject > android-13.0.0_r54 > libcore/ojluni/src/main/java/java/

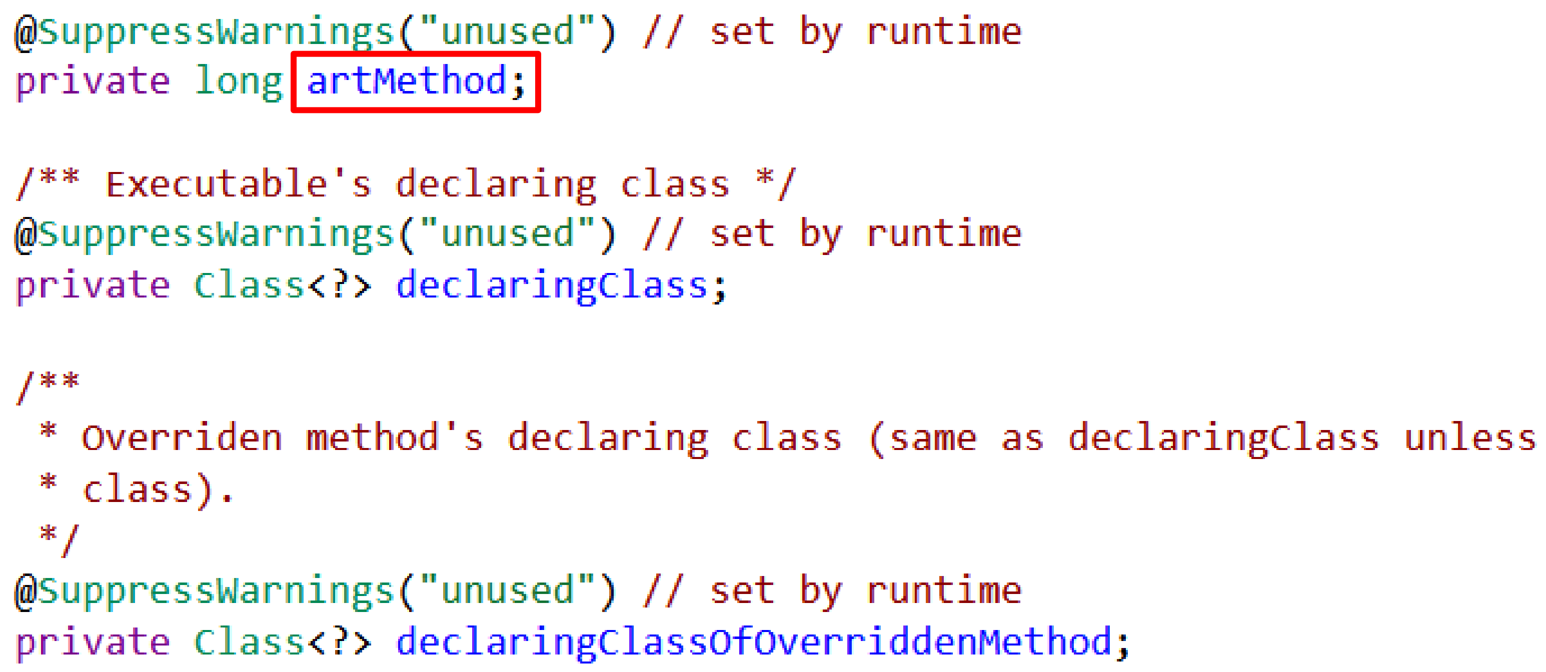
Files Outline

reflect
AccessibleObject.java
AnnotatedElement.java
Array.java
Constructor.java
Executable.java
Field.java
GenericArrayType.java
GenericDeclaration.java

Field.java

```
 * @author Kenneth Russell
 * @author Nakul Saraiya
 */
public final
class Field extends AccessibleObject i
    // Android-changed: Extensive modi
    // Android-changed: Many fields an
    // Android-removed: Type annotatio

    private int accessFlags;
    private Class<?> declaringClass;
    private int artFieldIndex;
    private int offset;
    private Class<?> type;
```

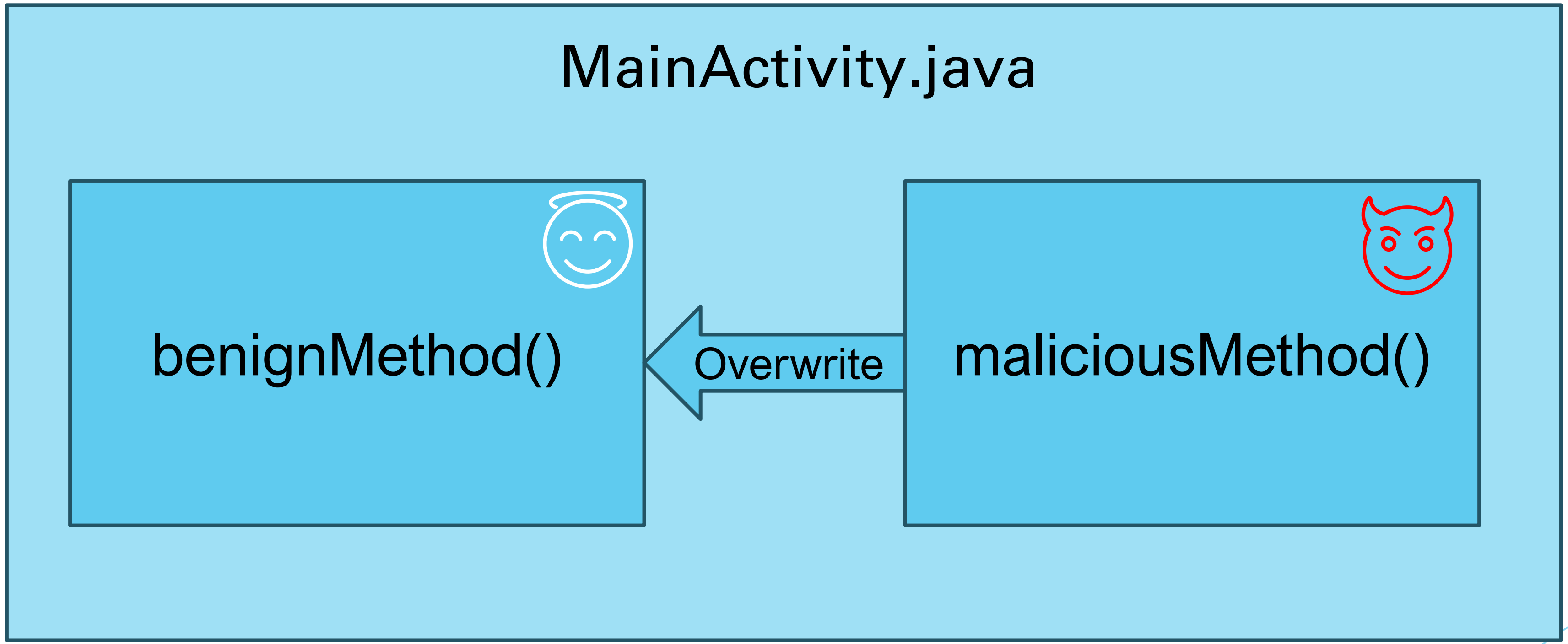
Target: Methods



```
@SuppressWarnings("unused") // set by runtime
private long artMethod;

/** Executable's declaring class */
@SuppressWarnings("unused") // set by runtime
private Class<?> declaringClass;

/**
 * Overriden method's declaring class (same as declaringClass unless
 * class).
 */
@SuppressWarnings("unused") // set by runtime
private Class<?> declaringClassOfOverriddenMethod;
```

Plan: Swap Methods at Runtime

Replace “Benign” with “Malicious”

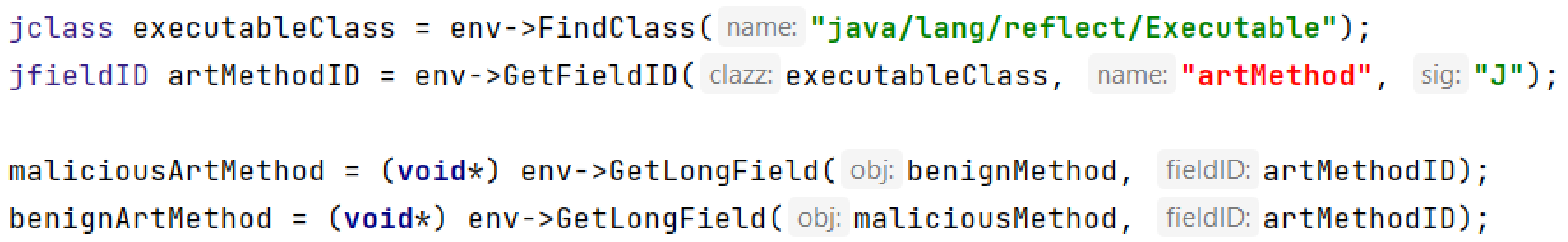


Intercepting Methods Via the JNI

Accessing Top-Level Java

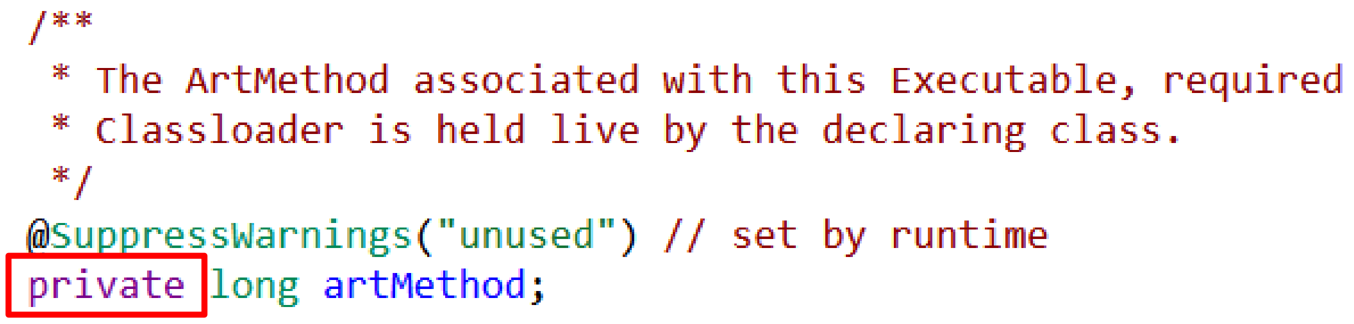
- Use the Java Native Interface
- Retrieve both method objects
- Load artMethod field

JNI Intercepting



```
jclass executableClass = env->FindClass( name: "java/lang/reflect/Executable");
jfieldID artMethodID = env->GetFieldID( clazz: executableClass,  name: "artMethod",  sig: "J");

maliciousArtMethod = (void*) env->GetLongField( obj: benignMethod,  fieldID: artMethodID);
benignArtMethod = (void*) env->GetLongField( obj: maliciousMethod,  fieldID: artMethodID);
```

artMethod is a private field ☹



```
/**
 * The ArtMethod associated with this Executable, required
 * Classloader is held live by the declaring class.
 */
@SuppressWarnings("unused") // set by runtime
private long artMethod;
```

Or is it?

The JNI doesn’t respect access modifiers lol

What if I overwrite the field?

```
benignMethod = maliciousMethod;
```

Test Time

We need to dive deeper.

Understanding Native Structures

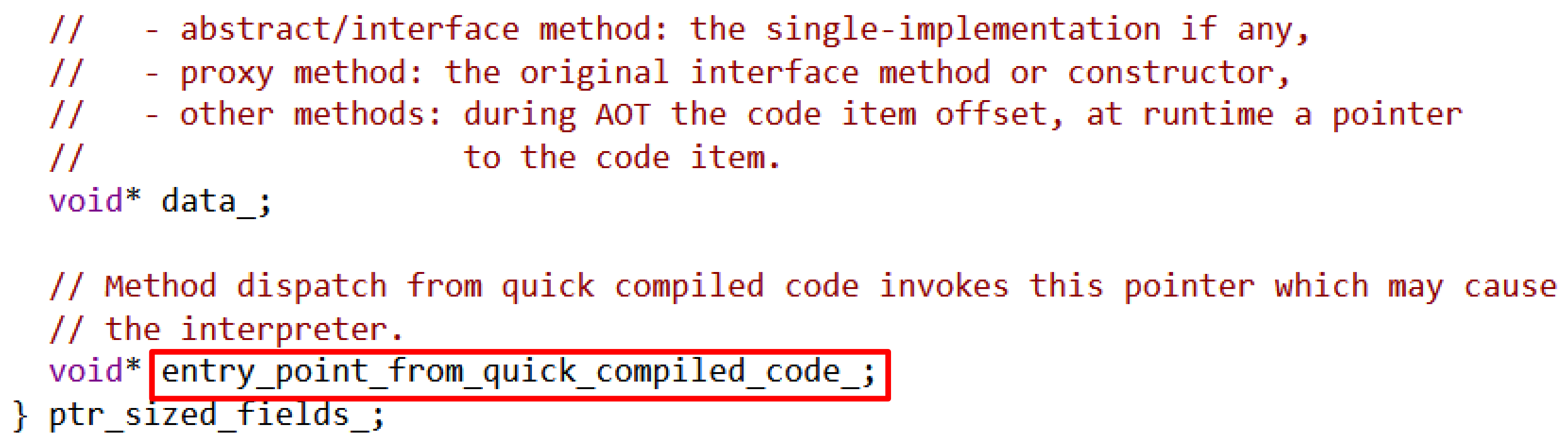
ArtMethod Structure

- artMethod field is a pointer
- Points to native structure
- Representation of ArtMethod

ArtMethod Native Implementation

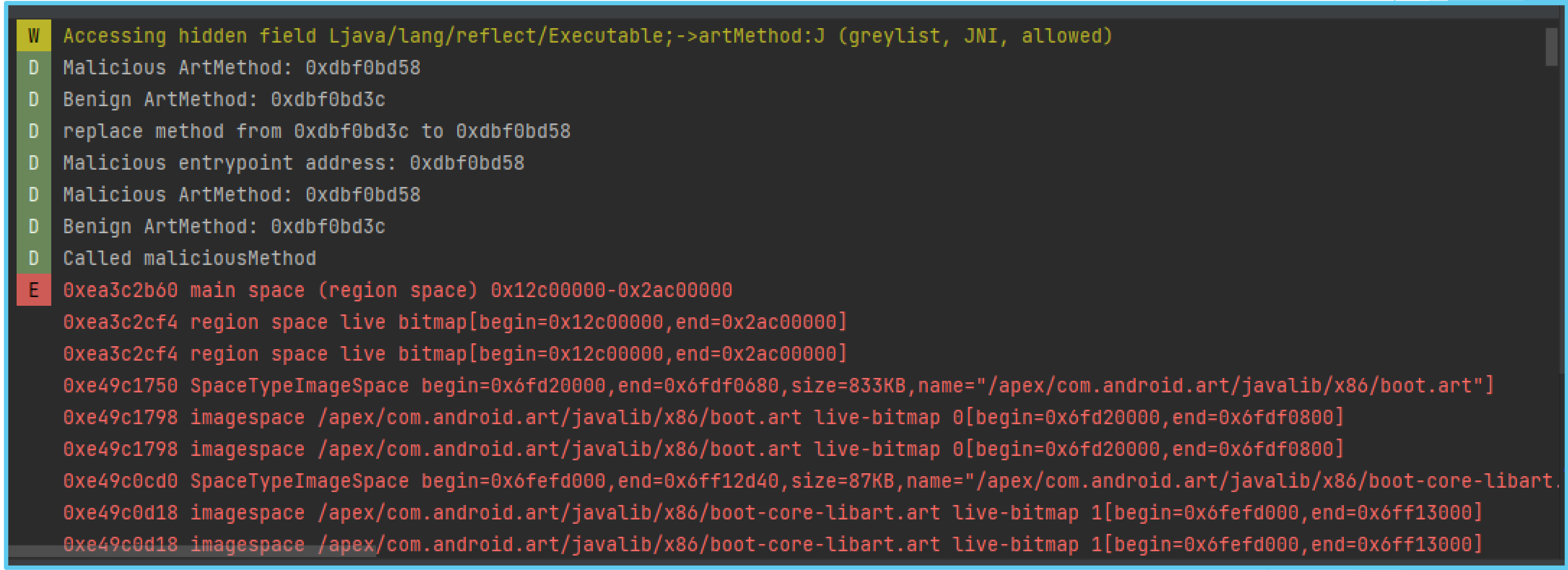
- Declared inside art_method.h
- Contains multiple method-related variables

Target: ArtMethod Entrypoint



```
    //   - abstract/interface method: the single-implementation if any,
    //   - proxy method: the original interface method or constructor,
    //   - other methods: during AOT the code item offset, at runtime a pointer
    //                    to the code item.
    void* data_;

    // Method dispatch from quick compiled code invokes this pointer which may cause
    // the interpreter.
    void* entry_point_from_quick_compiled_code_;
  } ptr_sized_fields_;
```

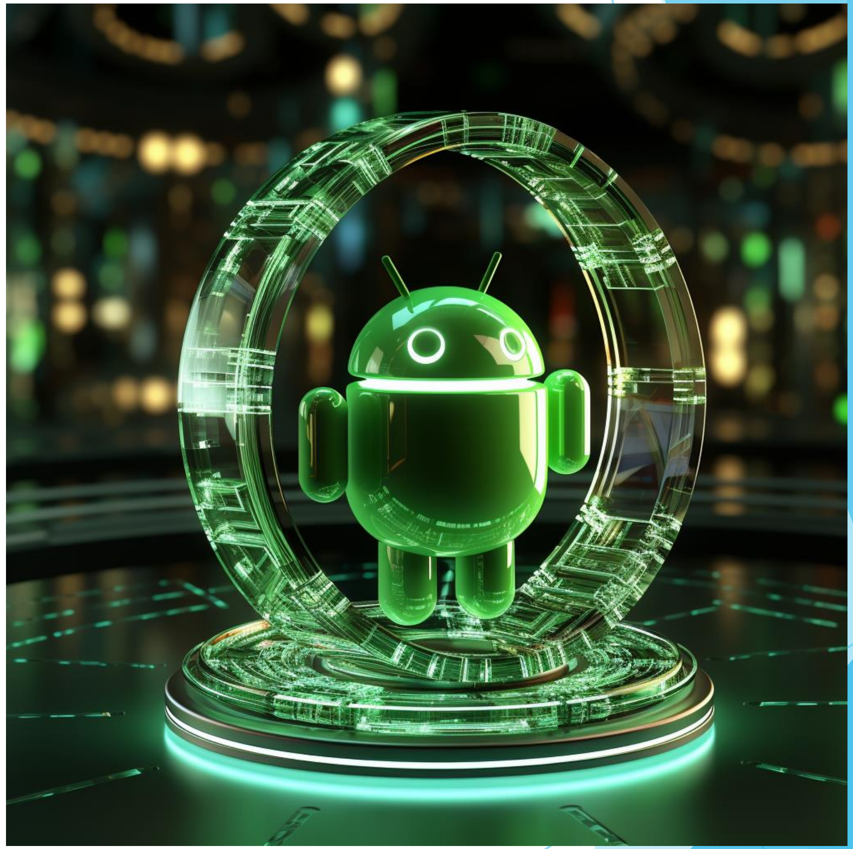
And we cause repeated crashes…



```
W  Accessing hidden field Ljava/lang/reflect/Executable;->artMethod:J (greylist, JNI, allowed)
D  Malicious ArtMethod: 0xdbf0bd58
D  Benign ArtMethod: 0xdbf0bd3c
D  replace method from 0xdbf0bd3c to 0xdbf0bd58
D  Malicious entrypoint address: 0xdbf0bd58
D  Malicious ArtMethod: 0xdbf0bd58
D  Benign ArtMethod: 0xdbf0bd3c
D  Called maliciousMethod
E  0xea3c2b60 main space (region space) 0x12c00000-0x2ac00000
   0xea3c2cf4 region space live bitmap[begin=0x12c00000,end=0x2ac00000]
   0xea3c2cf4 region space live bitmap[begin=0x12c00000,end=0x2ac00000]
   0xe49c1750 SpaceTypeImageSpace begin=0x6fd20000,end=0x6fdf0680,size=833KB,name="/apex/com.android.art/javalib/x86/boot.art"]
   0xe49c1798 imagespace /apex/com.android.art/javalib/x86/boot.art live-bitmap 0[begin=0x6fd20000,end=0x6fdf0800]
   0xe49c1798 imagespace /apex/com.android.art/javalib/x86/boot.art live-bitmap 0[begin=0x6fd20000,end=0x6fdf0800]
   0xe49c0cd0 SpaceTypeImageSpace begin=0x6fefd000,end=0x6ff12d40,size=87KB,name="/apex/com.android.art/javalib/x86/boot-core-libart.
   0xe49c0d18 imagespace /apex/com.android.art/javalib/x86/boot-core-libart.art live-bitmap 1[begin=0x6fefd000,end=0x6ff13000]
   0xe49c0d18 imagespace /apex/com.android.art/javalib/x86/boot-core-libart.art live-bitmap 1[begin=0x6fefd000,end=0x6ff13000]
```

Trial and error

- Test every 2 bytes
- Swap 8-byte areas
- Try to guess what is a pointer



This is not working.

Bright Idea: Blindly overwrite data

```
memcpy(benignArtMethod, maliciousArtMethod, 64);
```

Test 2 Time

Discoveries

- Must have the same signature
- Must be static
- May be declared in separate classes
- May have different functionality
- Works in application context

Increasing Accuracy: Pinpointing Native Fields

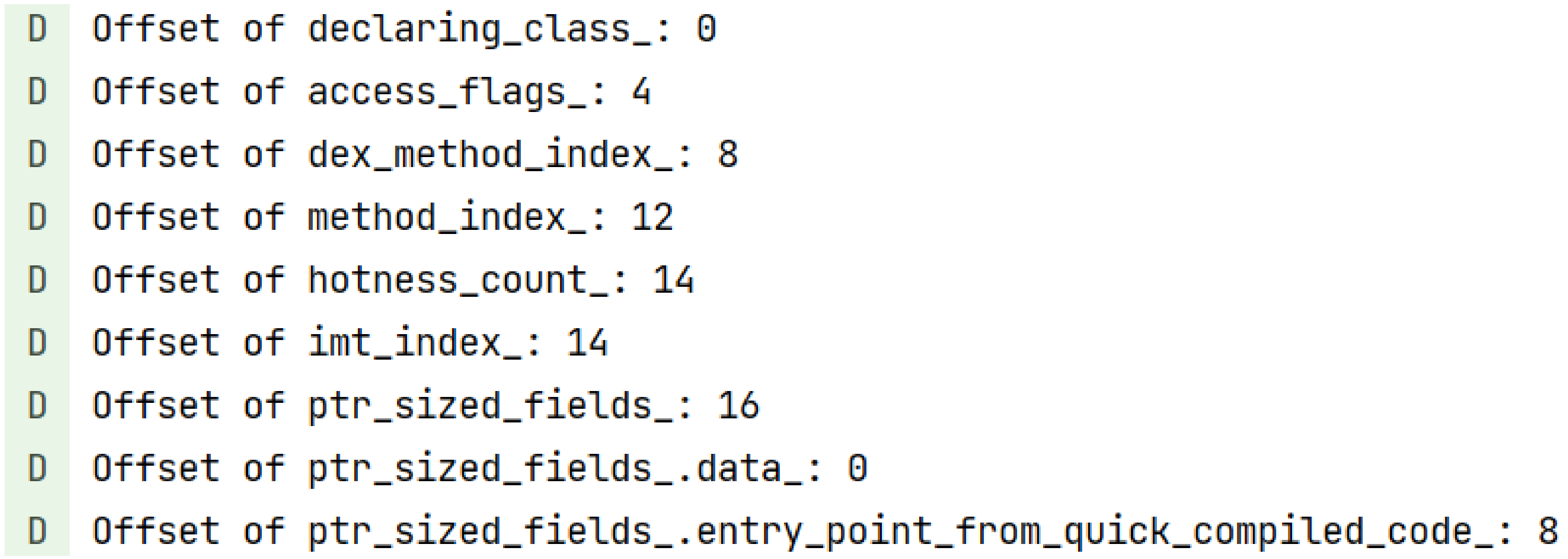
Offset Calculations

- Android source code uses offsets to locate member variables
- Used for getting and setting runtime values

Trick: Let it Calculate Itself

- Make a dummy ArtMethod class
- Only add member variables
- Let the program calculate itself

Printing Offsets from the App



```
D  Offset of declaring_class_: 0
D  Offset of access_flags_: 4
D  Offset of dex_method_index_: 8
D  Offset of method_index_: 12
D  Offset of hotness_count_: 14
D  Offset of imt_index_: 14
D  Offset of ptr_sized_fields_: 16
D  Offset of ptr_sized_fields_.data_: 0
D  Offset of ptr_sized_fields_.entry_point_from_quick_compiled_code_: 8
```

Time for some byte math!

Associating Runtime Bytes with Offsets

```
declaring_class_ (0)  access_flags_ (4)  dex_method_index_ (8)  method_index_ (12)
18 c8 03 13           09 00 38 10        1a 00 00 00            01 00
88 9c 67 71           09 00 38 10        a6 c1 00 00            08 00
```

```
hotness_count_ (14)  data_ (16)                 entry_point_from_quick_compiled_code_ (24)
ff ff                2c 77 80 b1 7e 74 00 00    90 03 b6 01 7c 74 00 00
ff ff                c8 f6 30 00 7c 74 00 00    90 03 b6 01 7c 74 00 00
```

Did it work?

Looks like assembly to me!

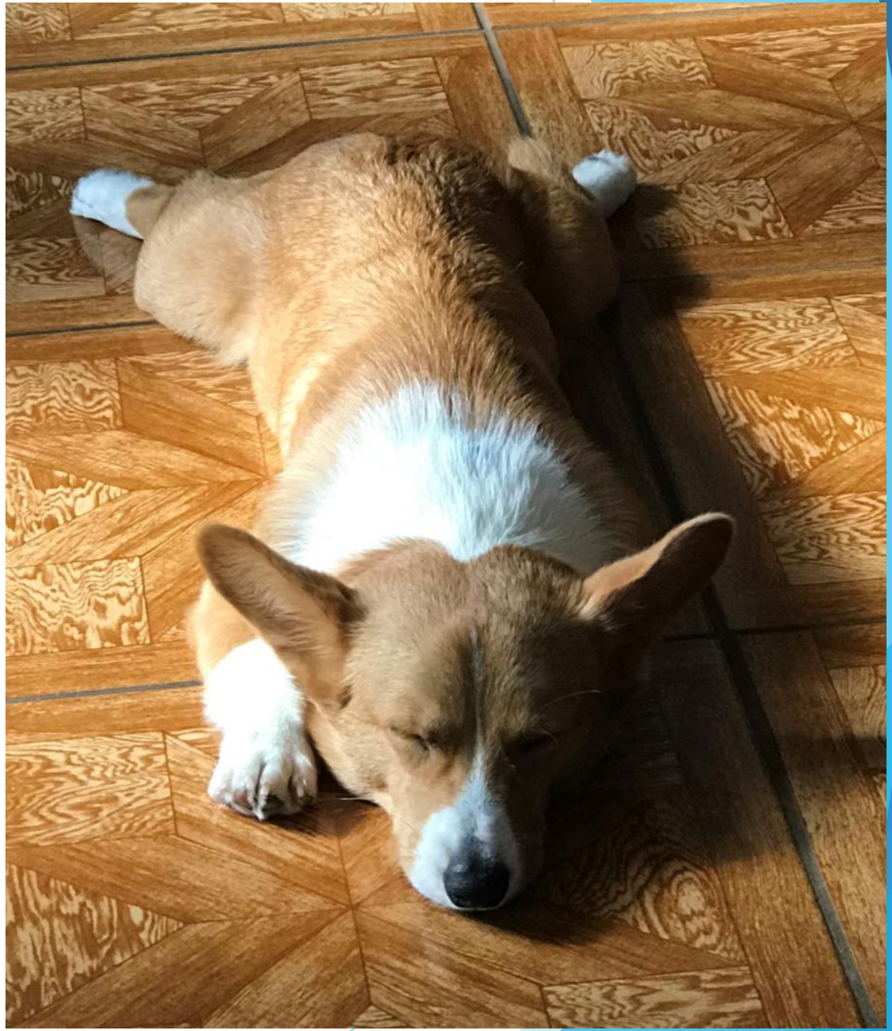
Disassembly:

```
0:  48 85 84 24 00 e0 ff    test   QWORD PTR [rsp-0x2000],rax
7:  ff
8:  41 57                   push   r15
a:  41 56                   push   r14
c:  41 55                   push   r13
e:  41 54                   push   r12
10: 55                      push   rbp
11: 53                      push   rbx
12: 48 83 ec 20             sub    rsp,0x20
16: 66 44 0f d6 24 24       movq   QWORD PTR [rsp],xmm12
```

Byte Overwriting Summary

Swapping full method	32 bytes in a 64-bit OS 24 bytes in a 32-bit OS
Replacing entrypoint	24th byte in a 64-bit OS 20th byte in a 32-bit OS

Calculations Complete



Let’s have some fun.

Idea: Replace developer APIs with malicious code

Target: Log.e()

- Logs errors to the terminal
- Used by many apps
- Make logging steal app data

Create Malicious Mirror Method

- Static method
- Matching signature
- Contains code to execute instead

Test Time

Try your hand at manipulating ART!

ARTful Open-Source Tool

- Built for manipulating Android13
- Library for swapping static methods at runtime

ARTful Capabilities

- Replace user methods or Android Framework methods
- Overwrite Android developer APIs
- Dummy ArtMethod class for printing offsets

Summarizing Results

ART Manipulation

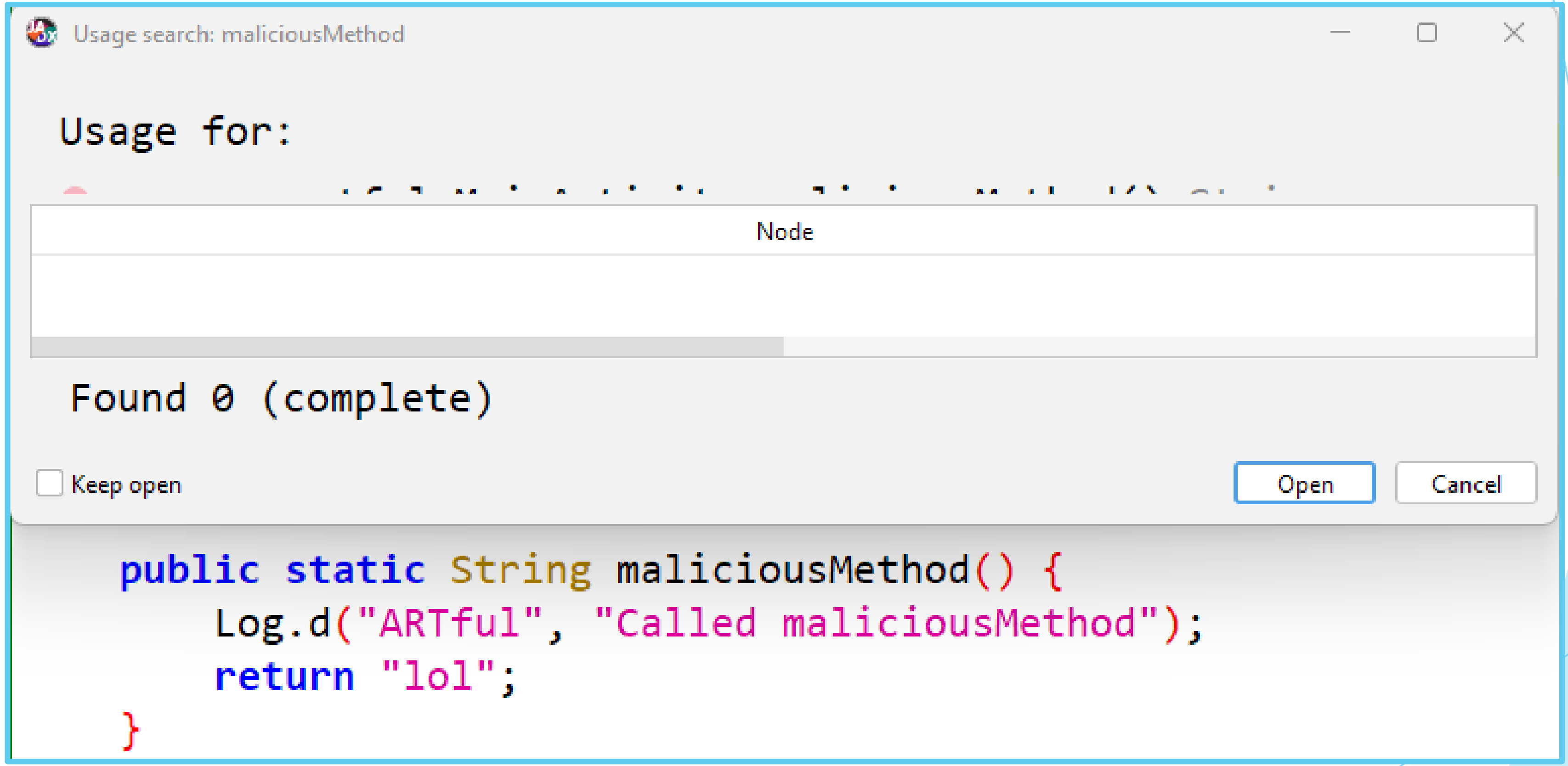
Challenges

- AOSP understanding
- Pointer math
- Differences in Android versions

Benefits

- Avoid standard Android APIs
- Execute unexpected code
- Thwart reverse engineering

No Method References



Usage search: maliciousMethod

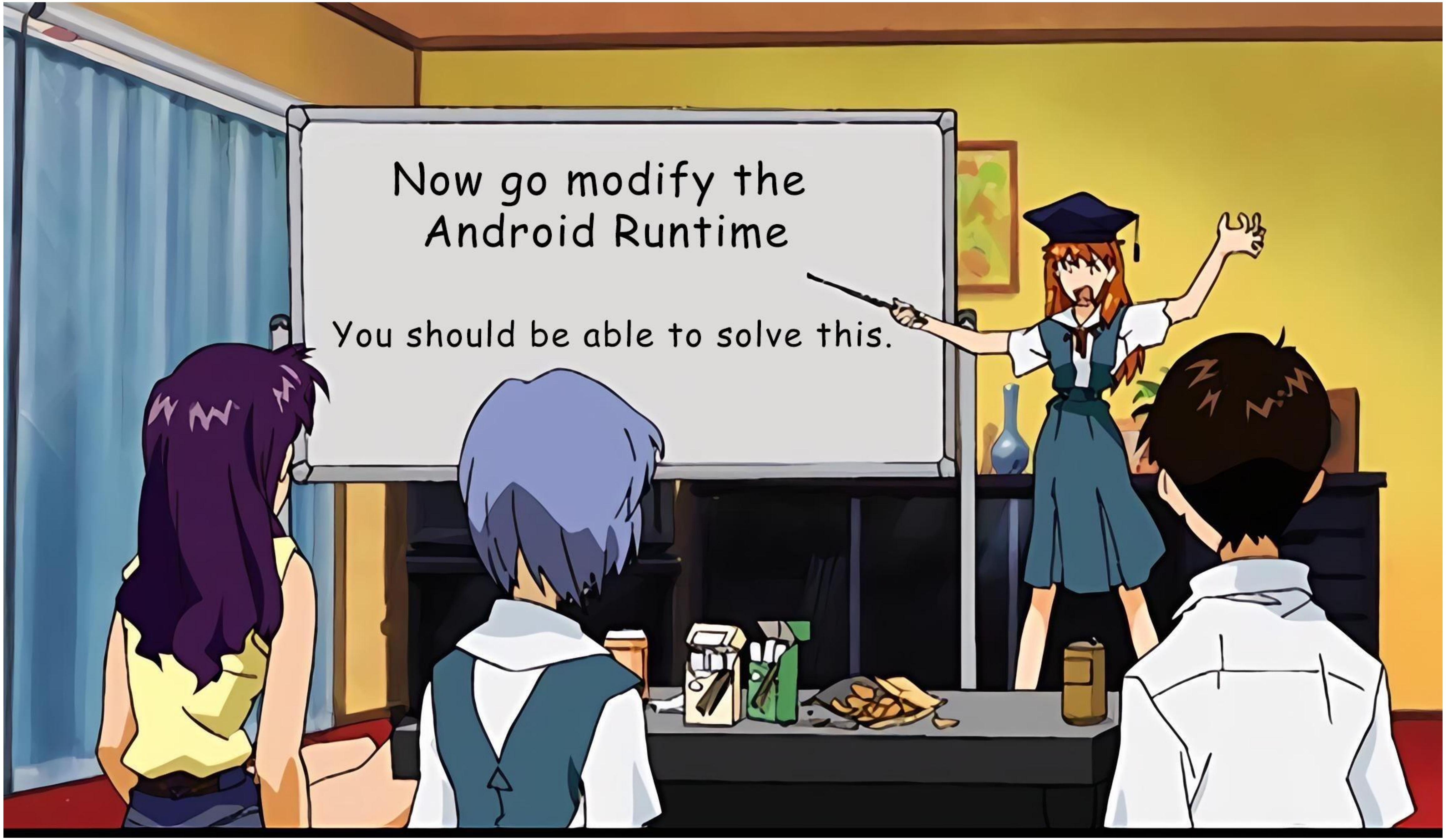
Usage for:

Found 0 (complete)

```
public static String maliciousMethod() {
    Log.d("ARTful", "Called maliciousMethod");
    return "lol";
}
```

No Calls to Hooked Android APIs

- DexClassLoader
- PathClassLoader
- ClassLoader.loadClass


Now go modify the Android Runtime
You should be able to solve this.

Thank you!

Bonus Section

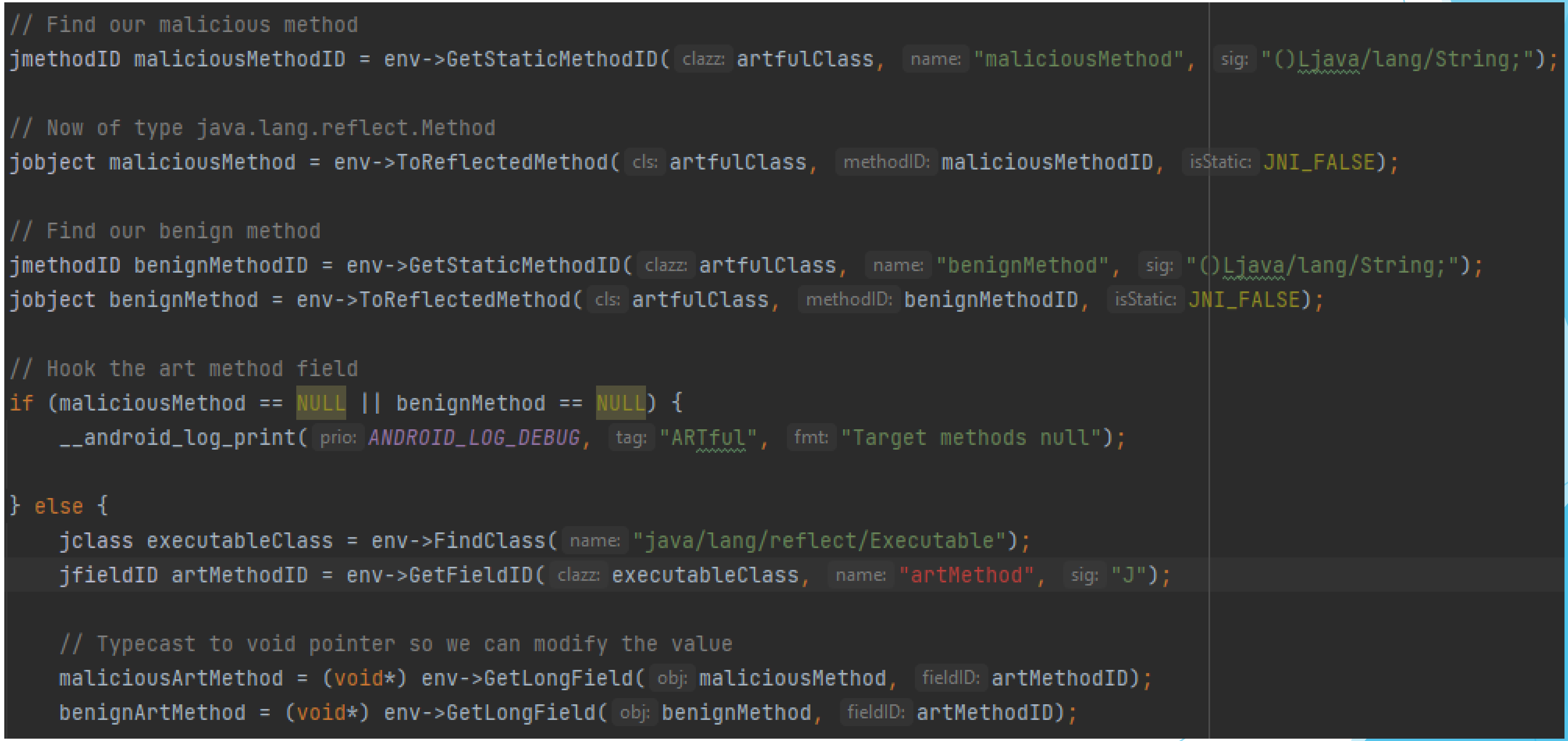
ARTful Tool

- LaurieWired ARTful Github Repo
  - https://github.com/LaurieWired/ARTful

Assembly References

- Online x86 Assembler
  - https://defuse.ca/online-x86-assembler.htm#disassembly2

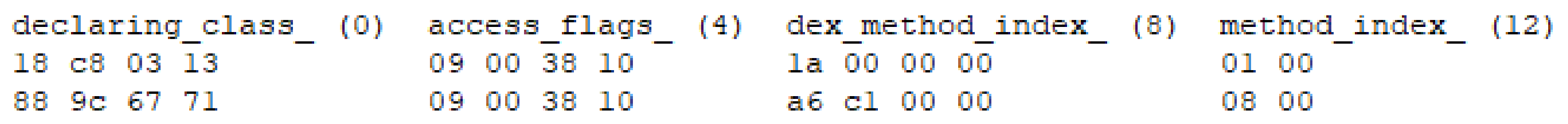
Intercept the artMethod Runtime Field



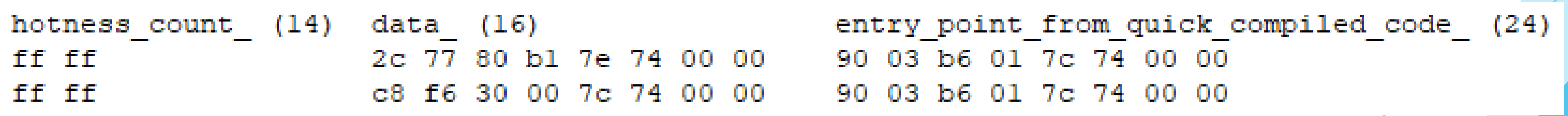
```
// Find our malicious method
jmethodID maliciousMethodID = env->GetStaticMethodID( clazz: artfulClass,  name: "maliciousMethod",  sig: "()Ljava/lang/String;");

// Now of type java.lang.reflect.Method
jobject maliciousMethod = env->ToReflectedMethod( cls: artfulClass,  methodID: maliciousMethodID,  isStatic: JNI_FALSE);

// Find our benign method
jmethodID benignMethodID = env->GetStaticMethodID( clazz: artfulClass,  name: "benignMethod",  sig: "()Ljava/lang/String;");
jobject benignMethod = env->ToReflectedMethod( cls: artfulClass,  methodID: benignMethodID,  isStatic: JNI_FALSE);

// Hook the art method field
if (maliciousMethod == NULL || benignMethod == NULL) {
    __android_log_print( prio: ANDROID_LOG_DEBUG,  tag: "ARTful",  fmt: "Target methods null");

} else {
    jclass executableClass = env->FindClass( name: "java/lang/reflect/Executable");
    jfieldID artMethodID = env->GetFieldID( clazz: executableClass,  name: "artMethod",  sig: "J");

    // Typecast to void pointer so we can modify the value
    maliciousArtMethod = (void*) env->GetLongField( obj: maliciousMethod,  fieldID: artMethodID);
    benignArtMethod = (void*) env->GetLongField( obj: benignMethod,  fieldID: artMethodID);
```

Calculating entrypoint offset

- Entrypoint should be PtrSizedFields + entry_point_from_quick_compiled_code_
- = 24



```
declaring_class_ (0)   access_flags_ (4)   dex_method_index_ (8)   method_index_ (12)
18 c8 03 13            09 00 38 10         1a 00 00 00             01 00
88 9c 67 71            09 00 38 10         a6 c1 00 00             08 00
```

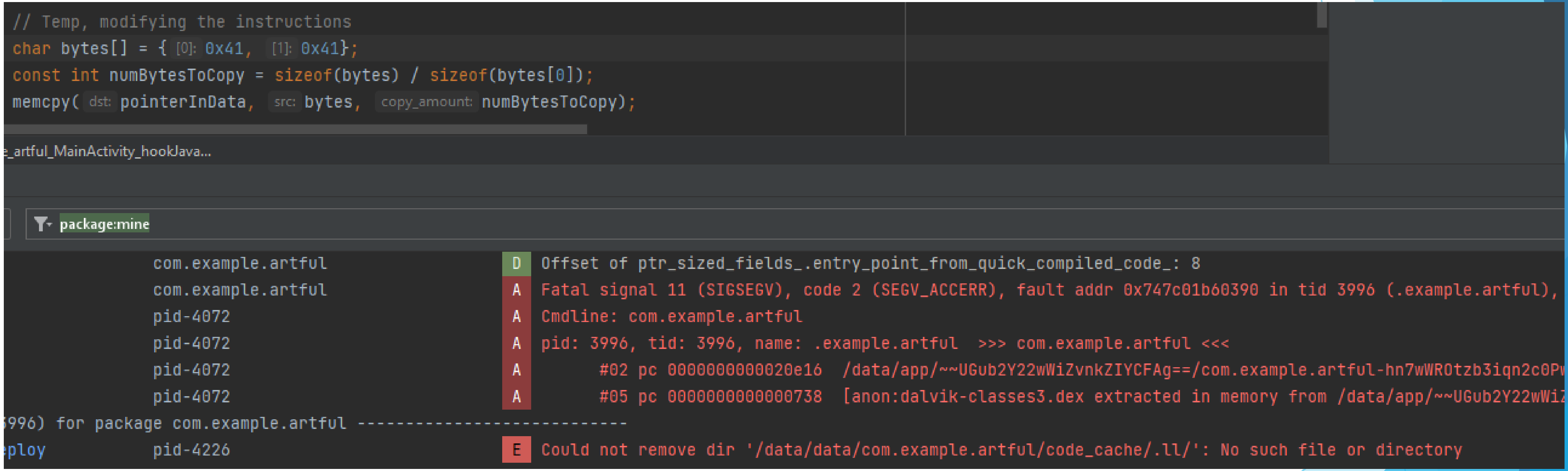


```
hotness_count_ (14)   data_ (16)                 entry_point_from_quick_compiled_code_ (24)
ff ff                 2c 77 80 b1 7e 74 00 00    90 03 b6 01 7c 74 00 00
ff ff                 c8 f6 30 00 7c 74 00 00    90 03 b6 01 7c 74 00 00
```

Printing Native Instructions (x64)

- Dereference pointer at offset 24
- Dump bytes pointed to by entrypoint
- Assemble to see if valid instructions

Back to crashing



```
// Temp, modifying the instructions
char bytes[] = { [0]: 0x41, [1]: 0x41};
const int numBytesToCopy = sizeof(bytes) / sizeof(bytes[0]);
memcpy( dst: pointerInData, src: bytes, copy_amount: numBytesToCopy);
```

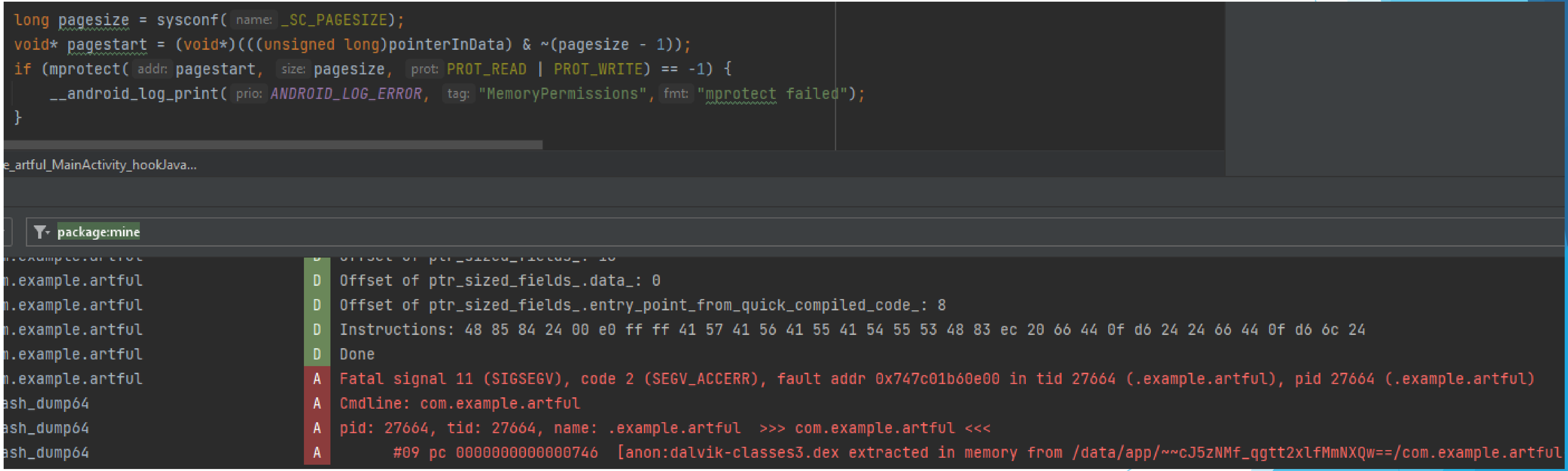
_artful_MainActivity_hookJava...

package:mine

```
            com.example.artful          D  Offset of ptr_sized_fields_.entry_point_from_quick_compiled_code_: 8
            com.example.artful          A  Fatal signal 11 (SIGSEGV), code 2 (SEGV_ACCERR), fault addr 0x747c01b60390 in tid 3996 (.example.artful),
            pid-4072                    A  Cmdline: com.example.artful
            pid-4072                    A  pid: 3996, tid: 3996, name: .example.artful  >>> com.example.artful <<<
            pid-4072                    A        #02 pc 0000000000020e16  /data/app/~~UGub2Y22wWiZvnkZIYCFAg==/com.example.artful-hn7wWROtzb3iqn2c0Pw
            pid-4072                    A        #05 pc 0000000000000738  [anon:dalvik-classes3.dex extracted in memory from /data/app/~~UGub2Y22wWiZ
996) for package com.example.artful ---------------------------
eploy       pid-4226                    E  Could not remove dir '/data/data/com.example.artful/code_cache/.ll/': No such file or directory
```

Memory is not writeable

- And we get a seg fault when trying to change the protections



```
long pagesize = sysconf( name: _SC_PAGESIZE);
void* pagestart = (void*)(((unsigned long)pointerInData) & ~(pagesize - 1));
if (mprotect( addr: pagestart,  size: pagesize,  prot: PROT_READ | PROT_WRITE) == -1) {
    __android_log_print( prio: ANDROID_LOG_ERROR,  tag: "MemoryPermissions",  fmt: "mprotect failed");
}
```

e_artful_MainActivity_hookJava...

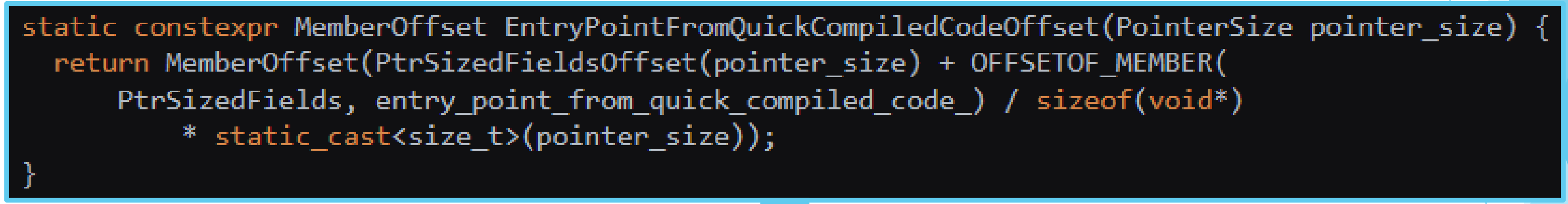
package:mine

```
n.example.artful          D  Offset of ptr_sized_fields_.data_: 0
n.example.artful          D  Offset of ptr_sized_fields_.entry_point_from_quick_compiled_code_: 8
n.example.artful          D  Instructions: 48 85 84 24 00 e0 ff ff 41 57 41 56 41 55 41 54 55 53 48 83 ec 20 66 44 0f d6 24 24 66 44 0f d6 6c 24
n.example.artful          D  Done
n.example.artful          A  Fatal signal 11 (SIGSEGV), code 2 (SEGV_ACCERR), fault addr 0x747c01b60e00 in tid 27664 (.example.artful), pid 27664 (.example.artful)
ash_dump64                A  Cmdline: com.example.artful
ash_dump64                A  pid: 27664, tid: 27664, name: .example.artful  >>> com.example.artful <<<
ash_dump64                A        #09 pc 0000000000000746  [anon:dalvik-classes3.dex extracted in memory from /data/app/~~cJ5zNMf_qgtt2xlfMmNXQw==/com.example.artful
```

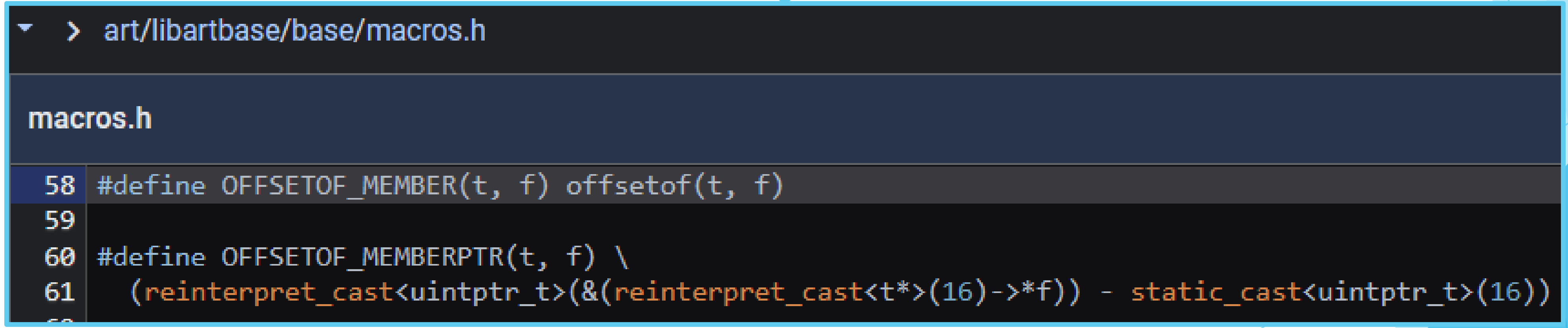
Allocate Native Instructions

- Still have possibilities
- Write new native instructions
- Point entrypoint to those instructions
- Avoid having methods on disk

Android Source Calculations



```
static constexpr MemberOffset EntryPointFromQuickCompiledCodeOffset(PointerSize pointer_size) {
  return MemberOffset(PtrSizedFieldsOffset(pointer_size) + OFFSETOF_MEMBER(
      PtrSizedFields, entry_point_from_quick_compiled_code_) / sizeof(void*)
          * static_cast<size_t>(pointer_size));
}
```



art/libartbase/base/macros.h

macros.h

```
#define OFFSETOF_MEMBER(t, f) offsetof(t, f)

#define OFFSETOF_MEMBERPTR(t, f) \
  (reinterpret_cast<uintptr_t>(&(reinterpret_cast<t*>(16)->*f)) - static_cast<uintptr_t>(16))
```