Advanced Obfuscation of Constants in Android Applications

Alexei Khlebnikov

LVEE 2018

Obfuscation

- Hiding info in plain sight
- Making info difficult to interpret

Hiding in plain sight

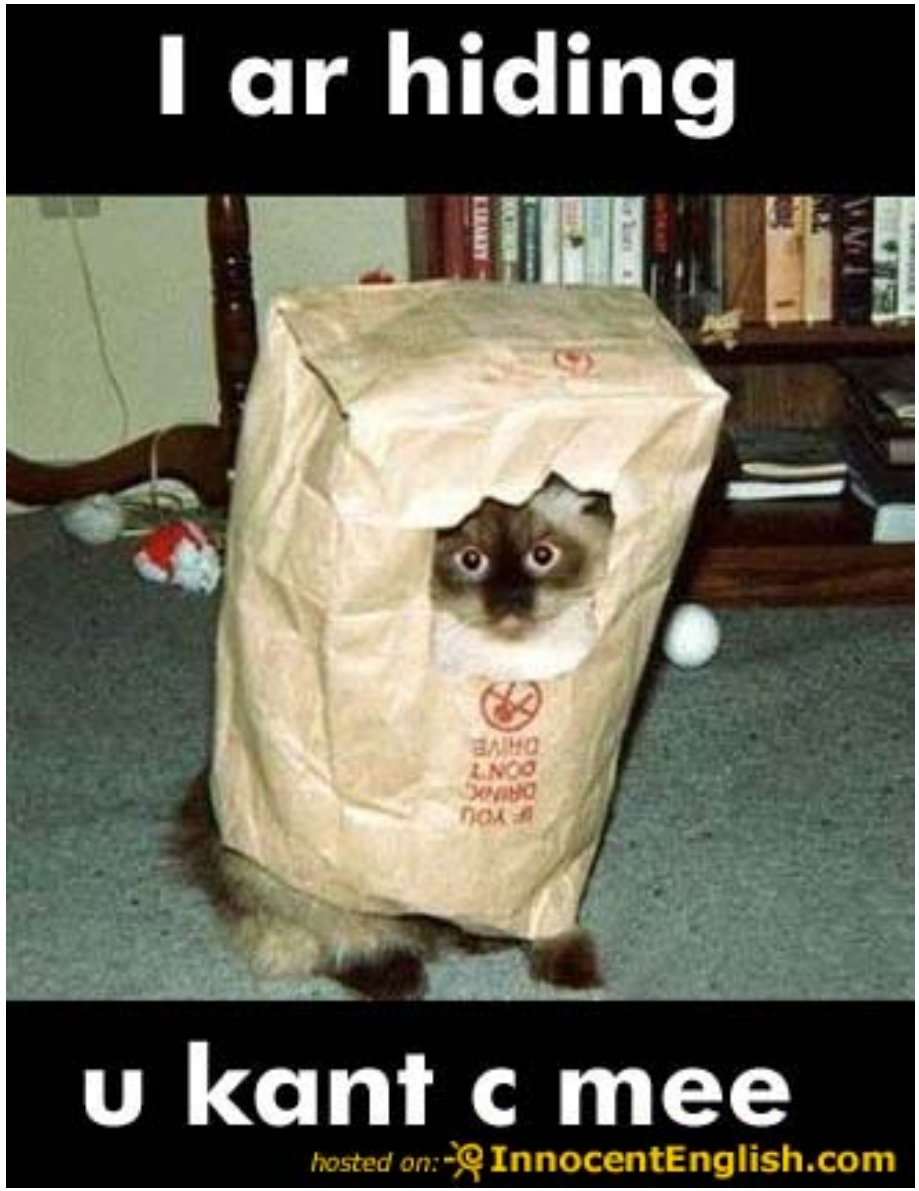

Figure 1:

Hiding in plain sight

Figure 2:

Hiding in plain sight

Figure 3:

Why people do obfuscation

- Making reverse engineering harder
- Harder $=>$ more expensive
- Reversing result is less worth the process
- Less people can do it
- Another target may be easier

Java is easily decompileable

- CFR
- Fernflower
- JADX
- JD
- JEB
- Krakatau
- Procyon
- Outdated: Candle, JAD

Other tools

- Apktool
- Dare
- dex2jar
- smali/baksmali

Decompiled dex, no obfuscation

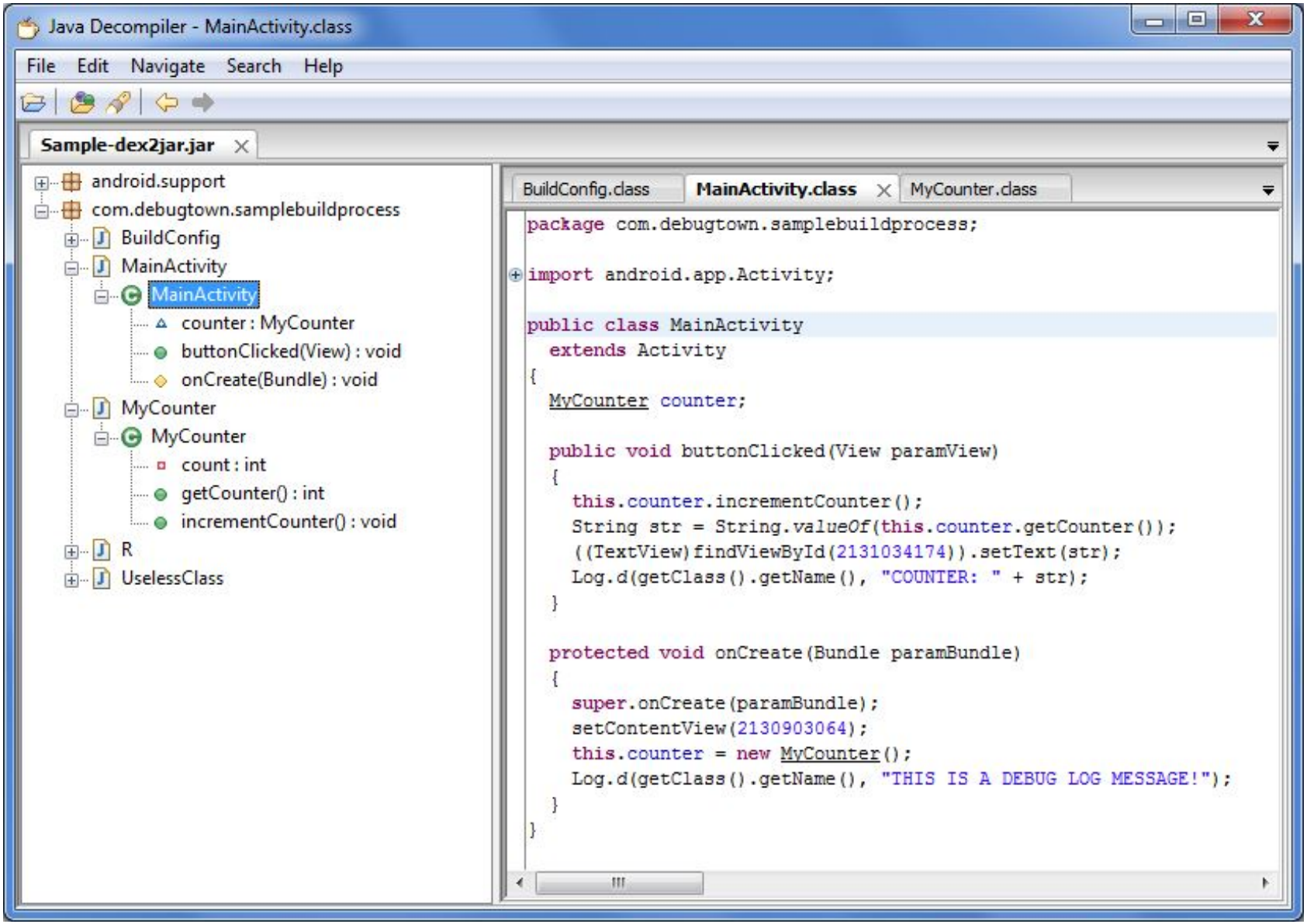

Figure 4:

Decompiled dex, ProGuard obfuscation

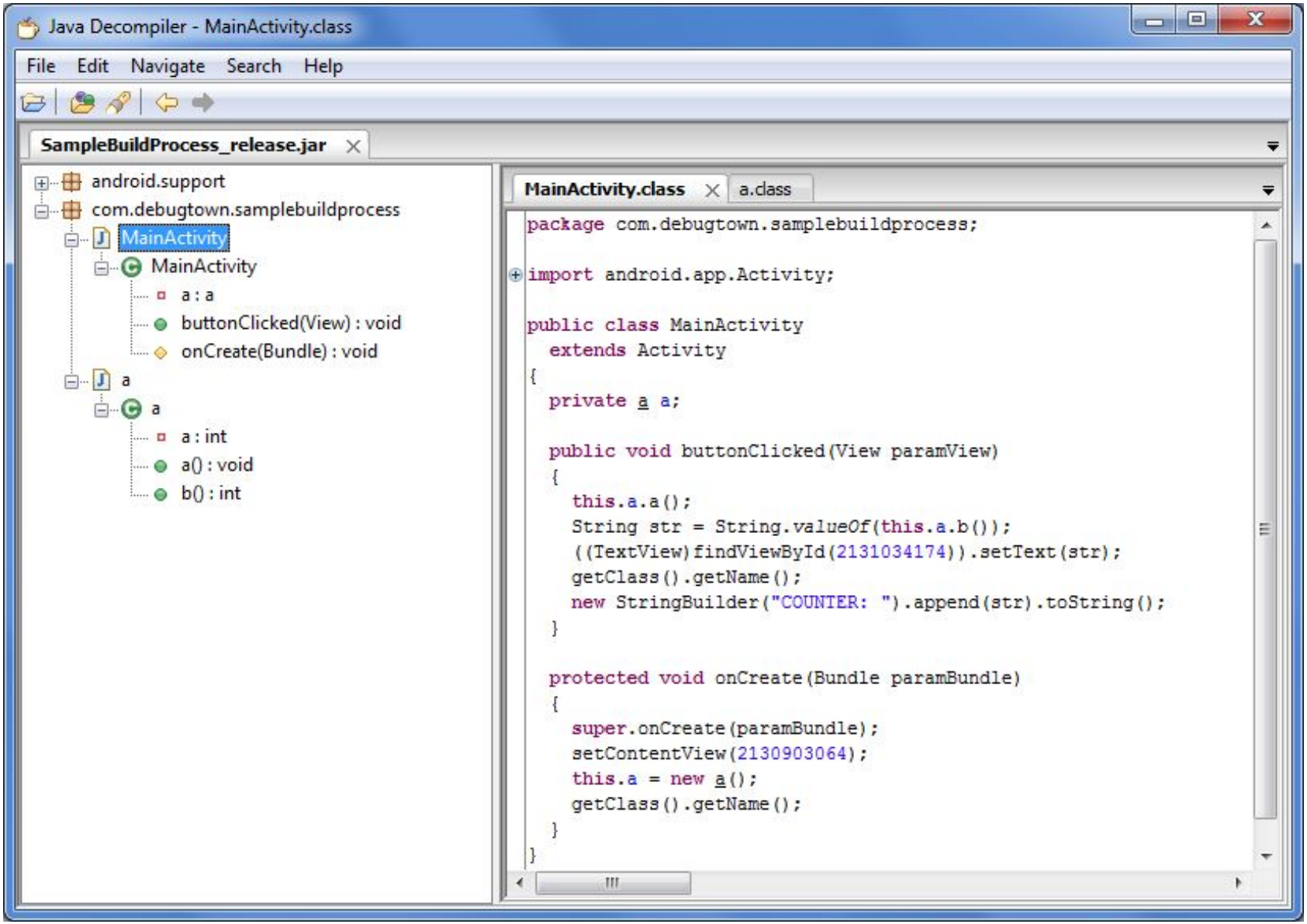

Figure 5:

Sensitive strings

- Secret keys/tokens/passwords
- Certificates
- Integrity hashes/digests
- Protocol keywords
- Older protocol versions
- Names of encryption methods
- Names of other used algorithms
- Names of used libraries
- URLs (for ex. to the licensing server)
- Bank account numbers

Unobfuscated string

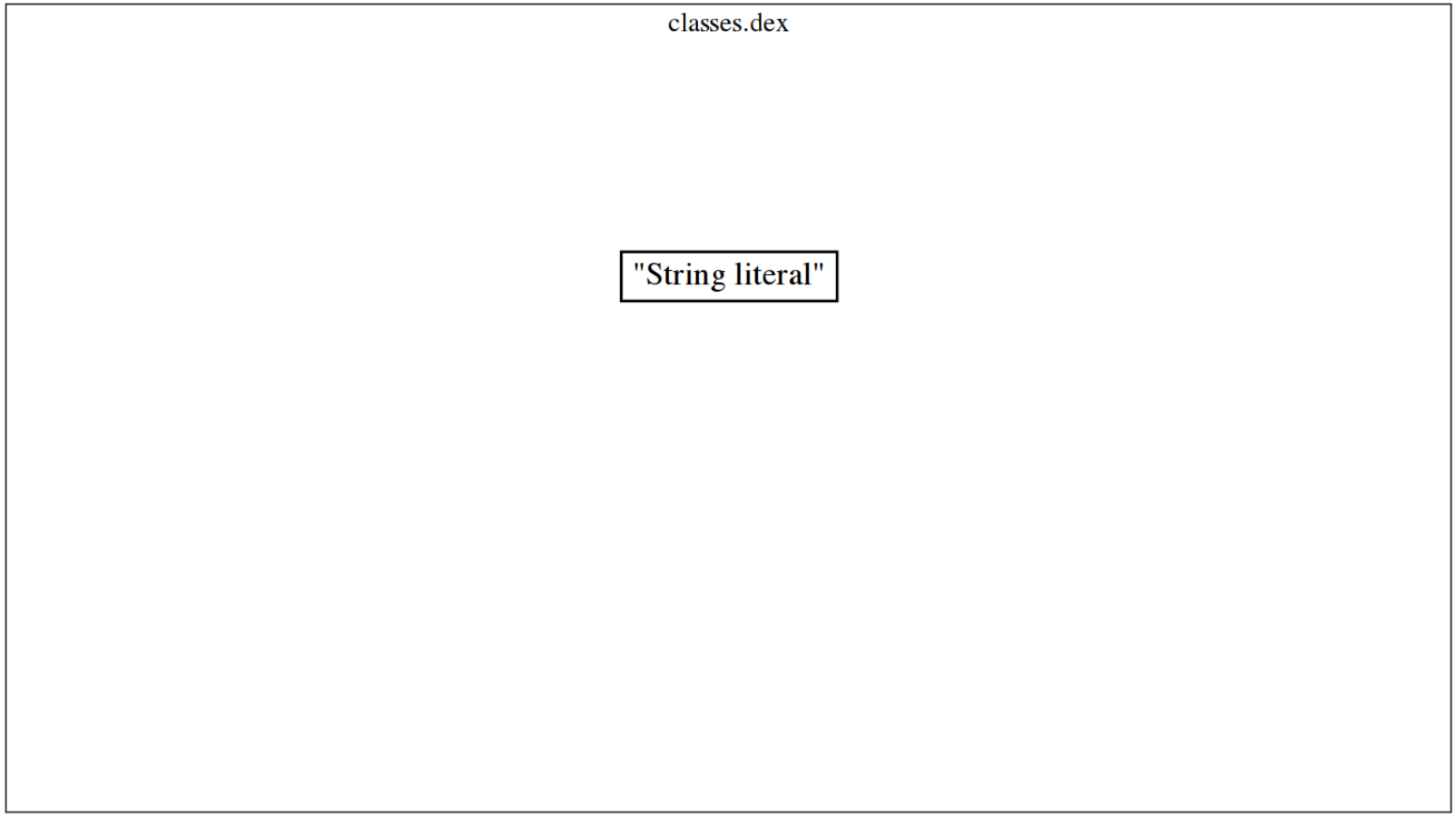

Figure 6:

String obfuscation, Java-only

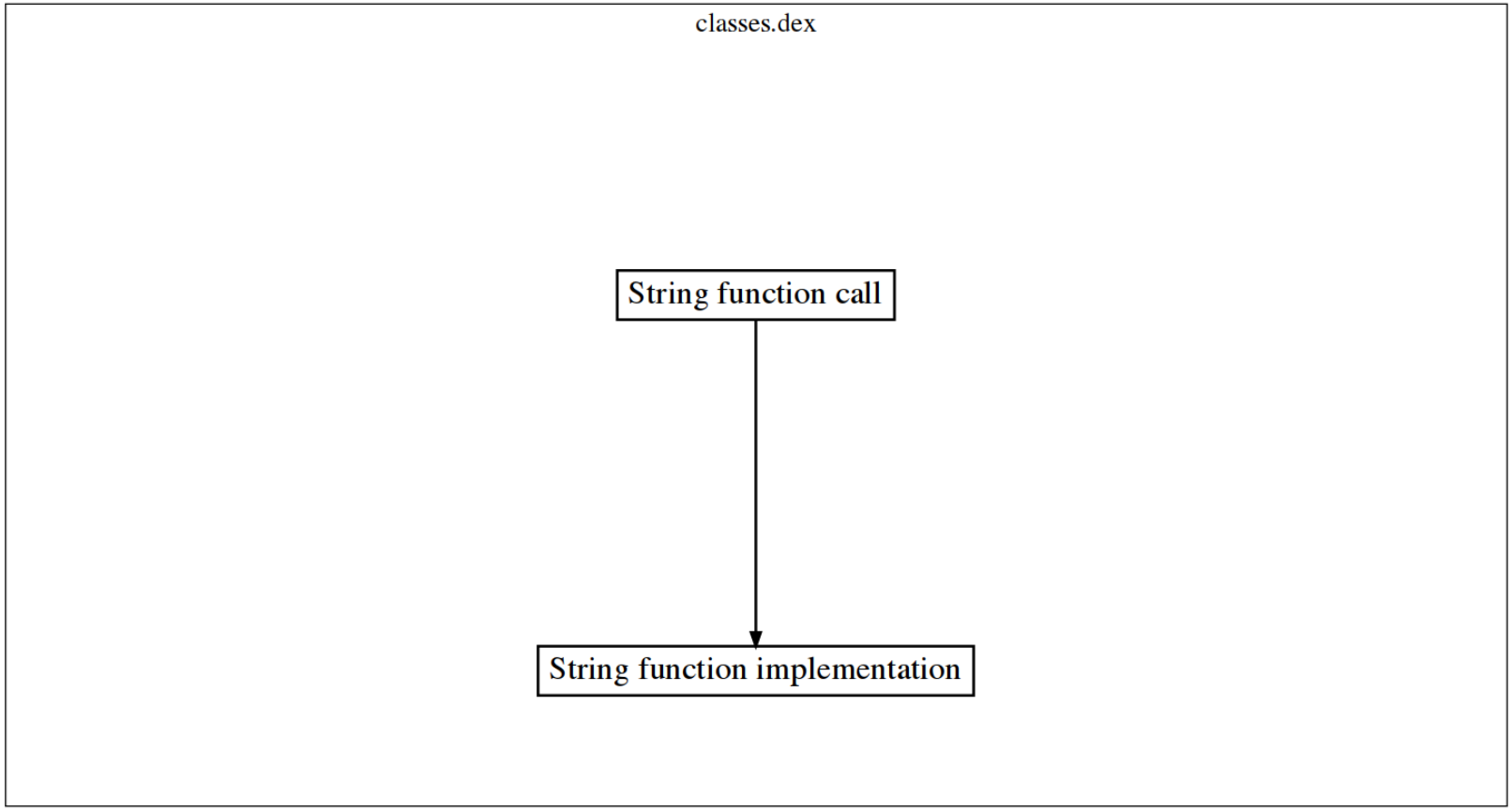

Figure 7:

Unobfuscated string

```
public class Class1
{
    private final static String secret = "supersecret";
};
```

String obfuscation, Java-only

```
public class Class1
{
    private static final byte[] string_data = new byte[] {
        110, -49, 71, -112, 33, -6, -12, 12, -25, -8, -33,
        47, 17, -4, -82, 82, 4, -74, 33, -35, 18, 7, -25, 31
    };

    private static String string_function(int var0, int var1_1, int var2_2) {
        var2_2 = var2_2 * 4 + 4;
        int var7_3 = var0 * 3 + 83;
        int var6_4 = -1;
        byte[] var3_5 = string_data;
        ...
        do {
            ...
            var4_7[++var1_1] = (byte)var5_8;
            if (var1_1 == var8_6 - 1) {
                return new String(var4_7, 0);
            }
            var2_2 = var5_8;
            var5_8 = var3_5[var0];
            var7_3 = var0;
        } while (true);
    }
};
```

String obfuscation, Java-only

- Replace string by an obfuscating function
- Example: DexGuard - easily reversible
  - https://www.pnfsoftware.com/blog/a-look-inside-dexguard/
  - https://ajinabraham.com/blog/reversing-dexguard-string-encryption

String obfuscation, using JNI

- Move strings to a native library
- Possible to COMPLETELY remove string data from Java code!
- Reverse engineering of compiled C/C++ is hard

JNI

- Java Native Interface
- Java-to-C calls
- C-to-Java calls
- Possible to use with Android NDK

String obfuscation, using JNI

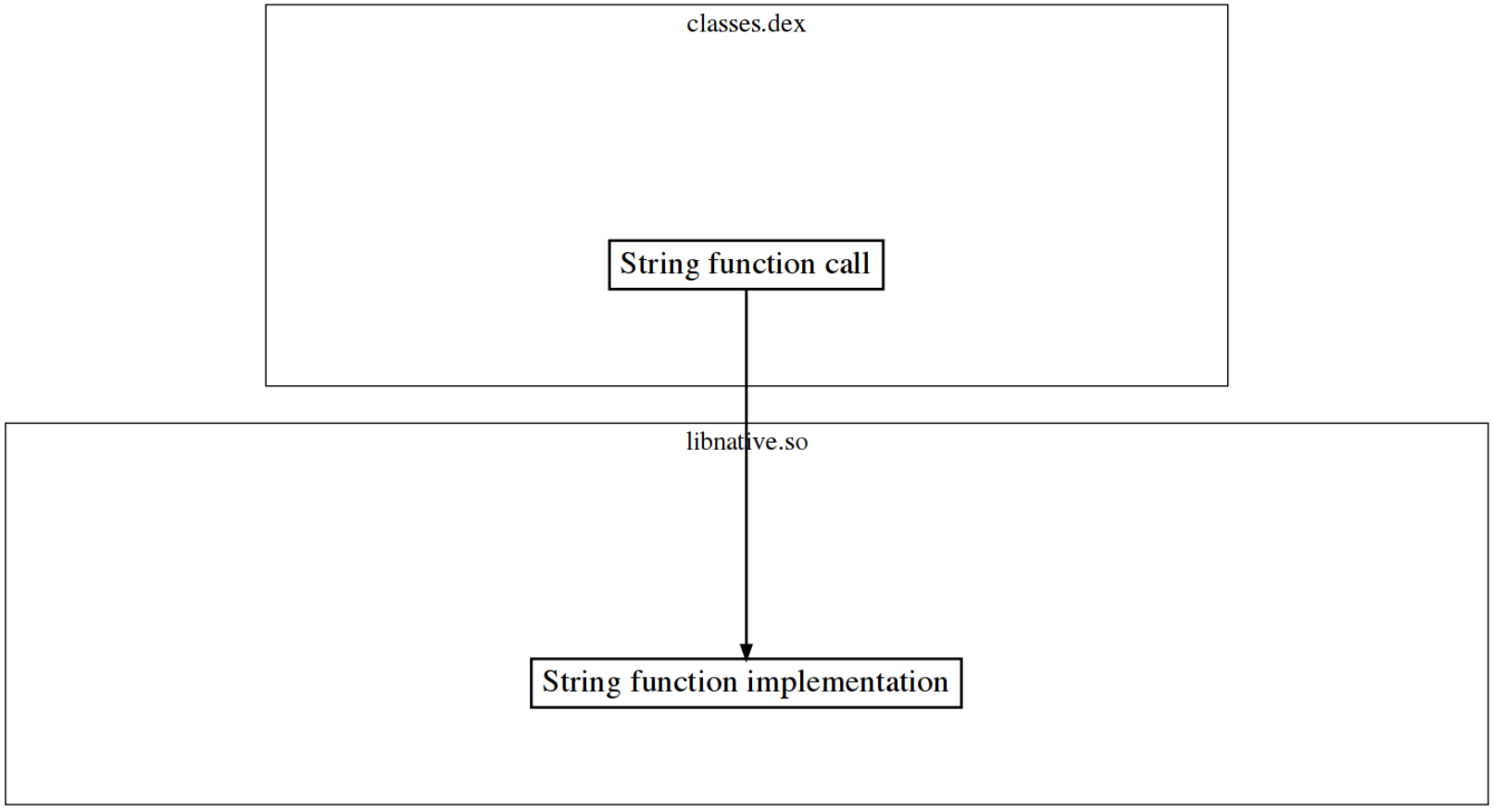

Figure 8:

String obfuscation, using JNI

```
public class Class1
{
    private final static String secret = getSecretString();

    private native String getSecretString();
};
```

String obfuscation, using JNI

```cpp
// File: provide_string.cpp

JNIEXPORT jstring JNICALL
Java_Class1_getSecretString(JNIEnv* env, jobject jthis)
{
    return env->NewStringUTF("supersecret");
}
```

String obfuscation, using JNI

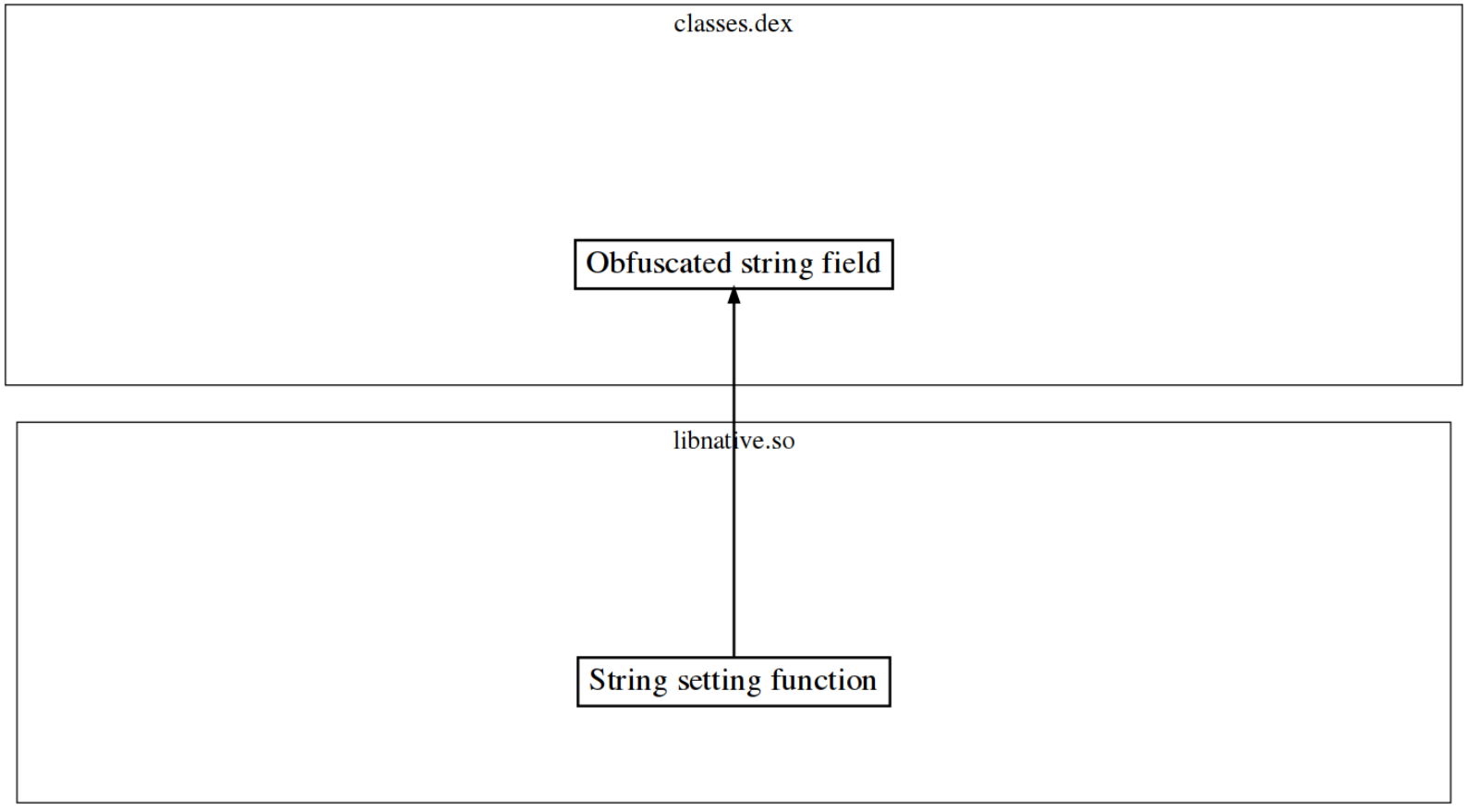

Figure 9:

String obfuscation, using JNI

```
public class Class1
{
    private final static String secret;
};
```

String obfuscation, using JNI

```cpp
// File: provide_string.cpp

void set_string(JNIEnv* env)
{
    auto clazz = env->FindClass("com/package/name/Class1");
    auto field = env->GetStaticFieldID(clazz, "secret", "[Ljava/lang/String;");
    auto java_string = env->NewStringUTF("supersecret");

    env->SetStaticObjectField(clazz, field, java_string);
}
```

O-LLVM

- LLVM fork
- Stands for: Obfuscator-LLVM
- Compiler that also obfuscates
- Compatible with Android NDK

Smali, what is it

- Android VM assembler
- Convertible from and to VM bytecode
- Assembling/disassembling possible with Apktool:

```
apktool decode app.apk
apktool build app_path
```

Smali, class example

```
.class interface abstract Lcalculator/Grapher;
.super Ljava/lang/Object;
.source "Grapher.java"


# static fields
.field public static final SCREENSHOT_DIR:Ljava/lang/String; = "/screenshots"


# virtual methods
.method public abstract captureScreenshot()Ljava/lang/String;
.end method

.method public abstract onPause()V
.end method

.method public abstract onResume()V
.end method

.method public abstract setFunction(Lorg/javia/arity/Function;)V
.end method
```

Smali, method examples

```
# direct methods
.method constructor <init>()V
    .locals 0

    .prologue
    .line 9
    invoke-direct {p0}, Landroid/text/Editable$Factory;-><init>()V

    return-void
.end method


# virtual methods
.method public newEditable(Ljava/lang/CharSequence;)Landroid/text/Editable;
    .locals 1
    .param p1, "source"    # Ljava/lang/CharSequence;

    .prologue
    .line 11
    new-instance v0, Lcalculator/CalculatorEditable;

    invoke-direct {v0, p1}, Lcalculator/CalculatorEditable;-><init>(Ljava/lang/

    return-object v0
.end method
```

Automating the process

1. Decode APK
   - apktool decode app.apk
2. Move string data from Smali code into native library
3. Encode rewritten Smali
   - apktool build app_path
4. ??? (Obligatory item before “PROFIT!”)
5. PROFIT!

Rewrite Smali ???

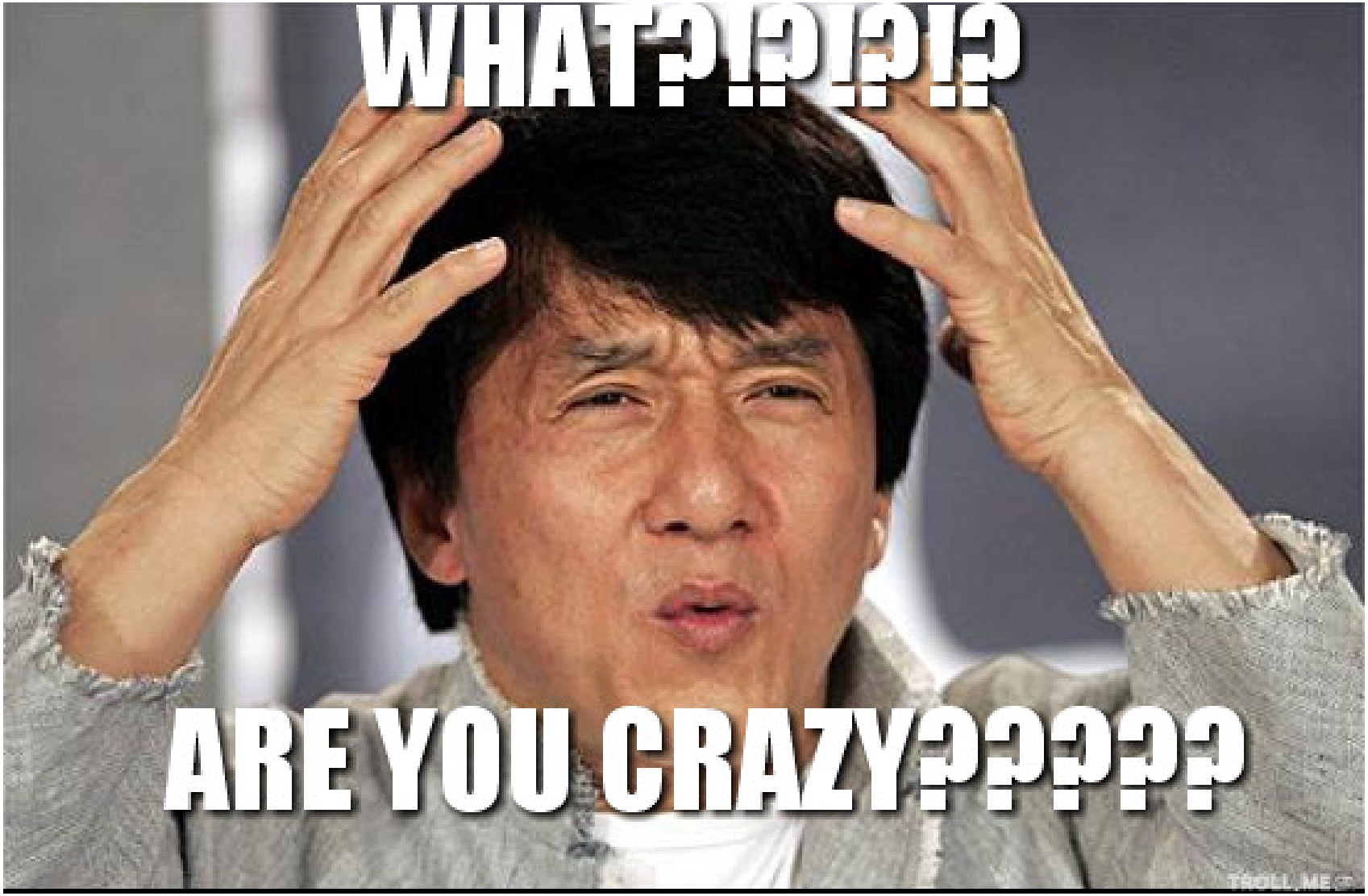

Figure 10:

String field, Java

```
public class Class1
{
    private final static String secret = "supersecret";
};
```

Unobfuscated string field

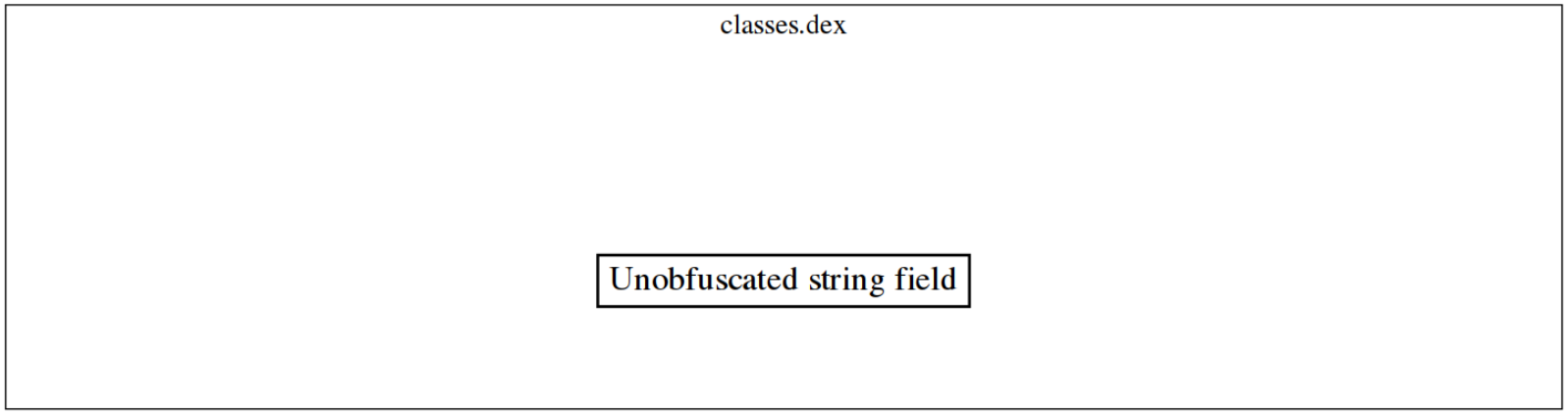

Figure 11:

Setting string field

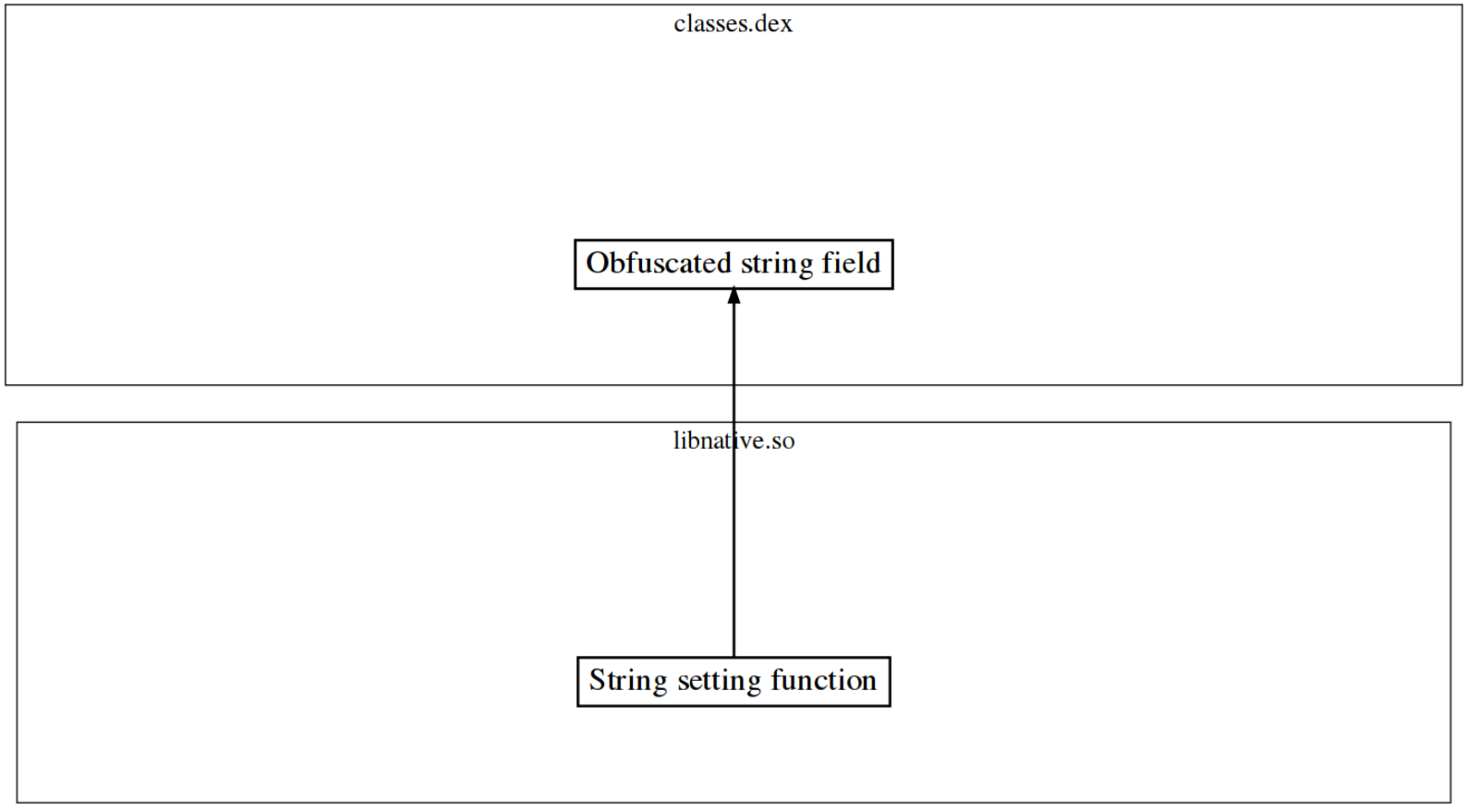

Figure 12:

String field, Smali

```
.class public LClass1;
.super Ljava/lang/Object;
.source "Class1.java"


# static fields
.field private static final secret:Ljava/lang/String; = "supersecret"


# direct methods
.method public constructor <init>()V
    .locals 0

    .prologue
    .line 1
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    return-void
.end method
```

String data removed, Smali

```
.class public LClass1;
.super Ljava/lang/Object;
.source "Class1.java"


# static fields
.field private static final secret:Ljava/lang/String; = null


# direct methods
.method public constructor <init>()V
    .locals 0

    .prologue
    .line 1
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    return-void
.end method
```

String data removed, Diff

```
--- Class1.smali
+++ Class1.smali.obfu
@@ -5,5 +5,5 @@

 # static fields
-.field private static final secret:Ljava/lang/String; = "supersecret"
+.field private static final secret:Ljava/lang/String; = null
```

String literal, Java

```
public class Class2
{
    void method1()
    {
        String secret = "supersecret";
    }
};
```

Function uses string literal

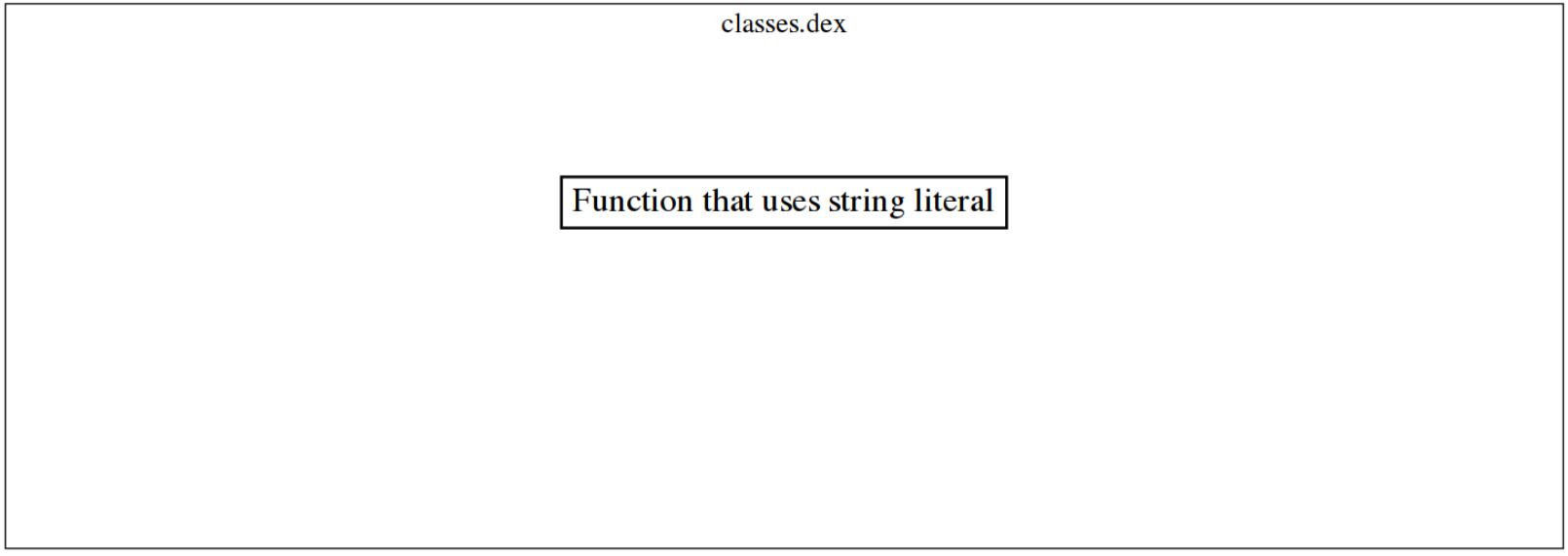

Figure 13:

Function uses obfuString field

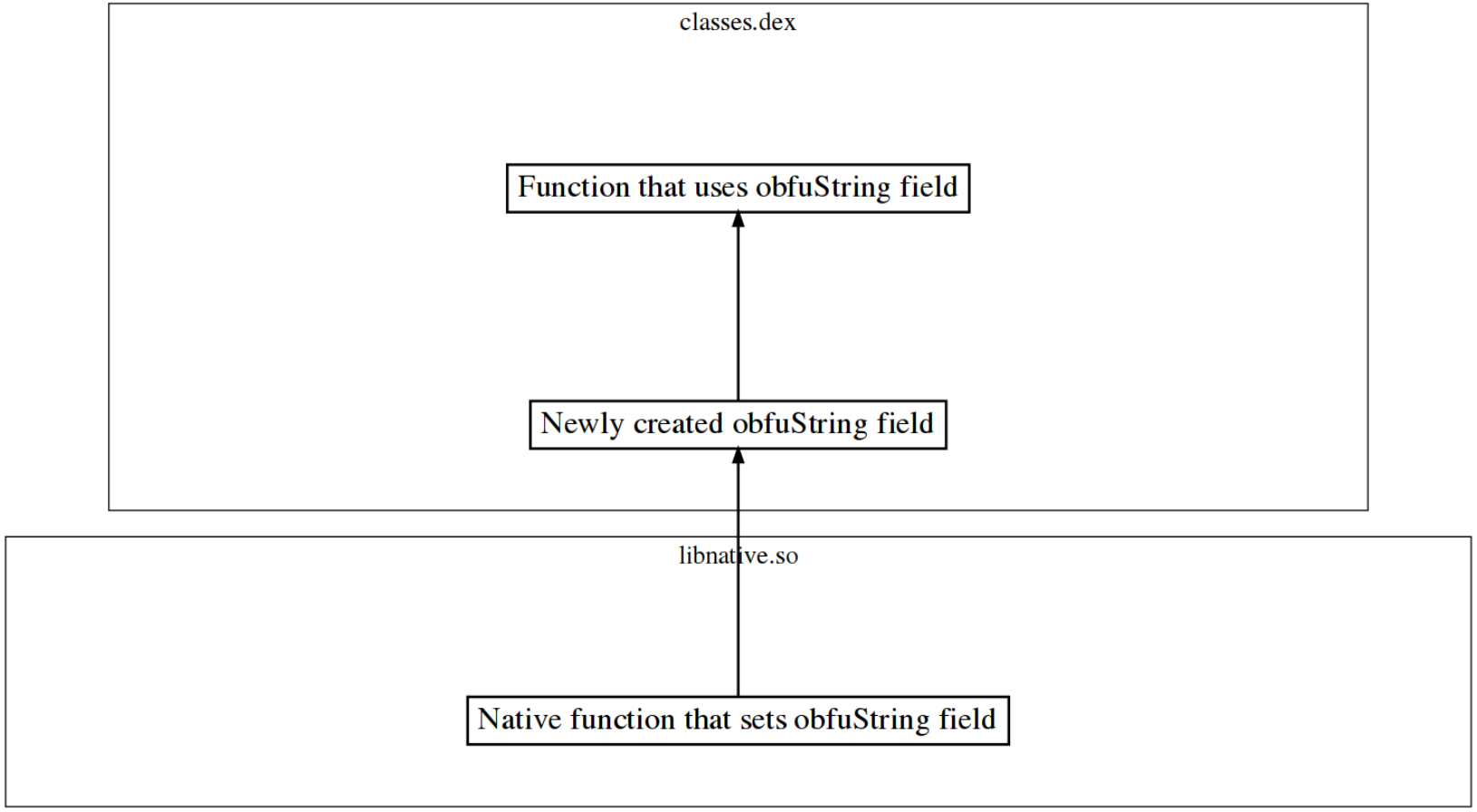

Figure 14:

String literal, Smali

```
# virtual methods
.method method1()V
    .locals 1

    .prologue
    .line 5
    const-string v0, "supersecret"

    .line 6
    return-void
.end method
```

String data removed, Smali

```
# static fields
.field private static final obfuString:Ljava/lang/String; = null


# virtual methods
.method method1()V
    .locals 1

    .prologue
    .line 5
    sget-object v0, LClass2;->obfuString:Ljava/lang/String;

    .line 6
    return-void
.end method
```

String data removed, Diff

```
--- Class2.smali
+++ Class2.smali.obfu
@@ -1,11 +1,15 @@
+# static fields
+.field private static final obfuString:Ljava/lang/String; = null
+
+
 # virtual methods
 .method method1()V
     .locals 1

     .prologue
     .line 5
-    const-string v0, "supersecret"
+    sget-object v0, LClass2;->obfuString:Ljava/lang/String;

     .line 6
     return-void
 .end method
```

Function uses string literal

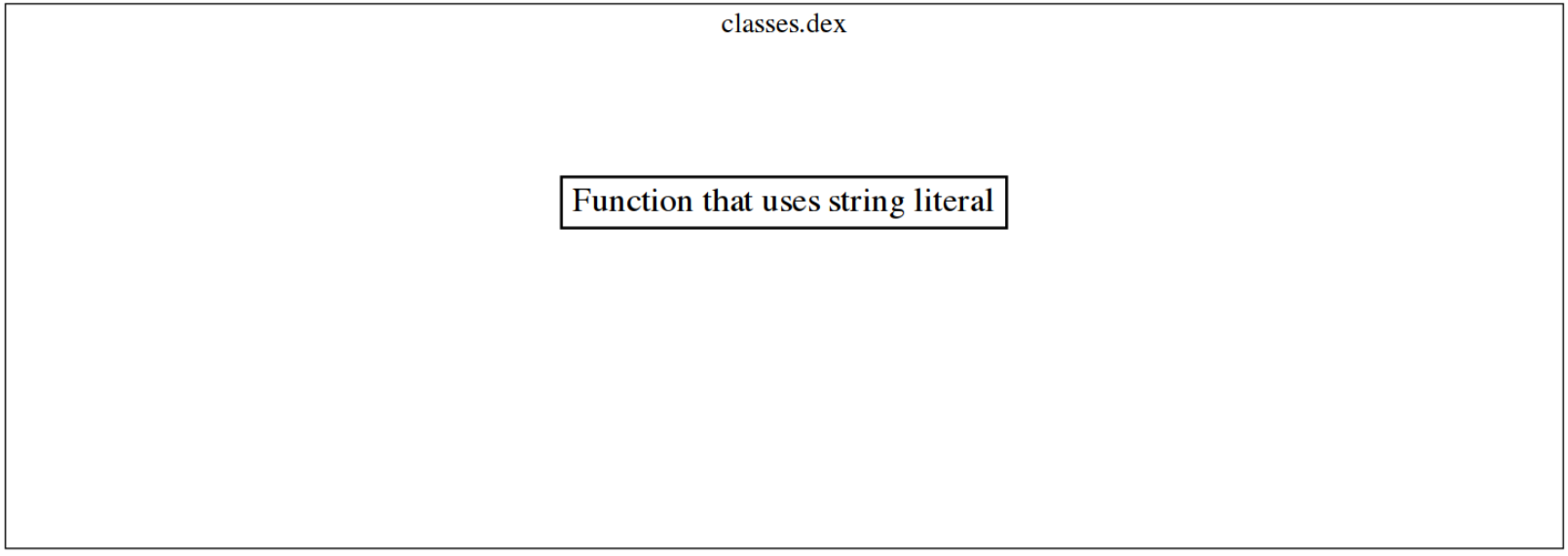

Figure 15:

Function uses getObfuString()

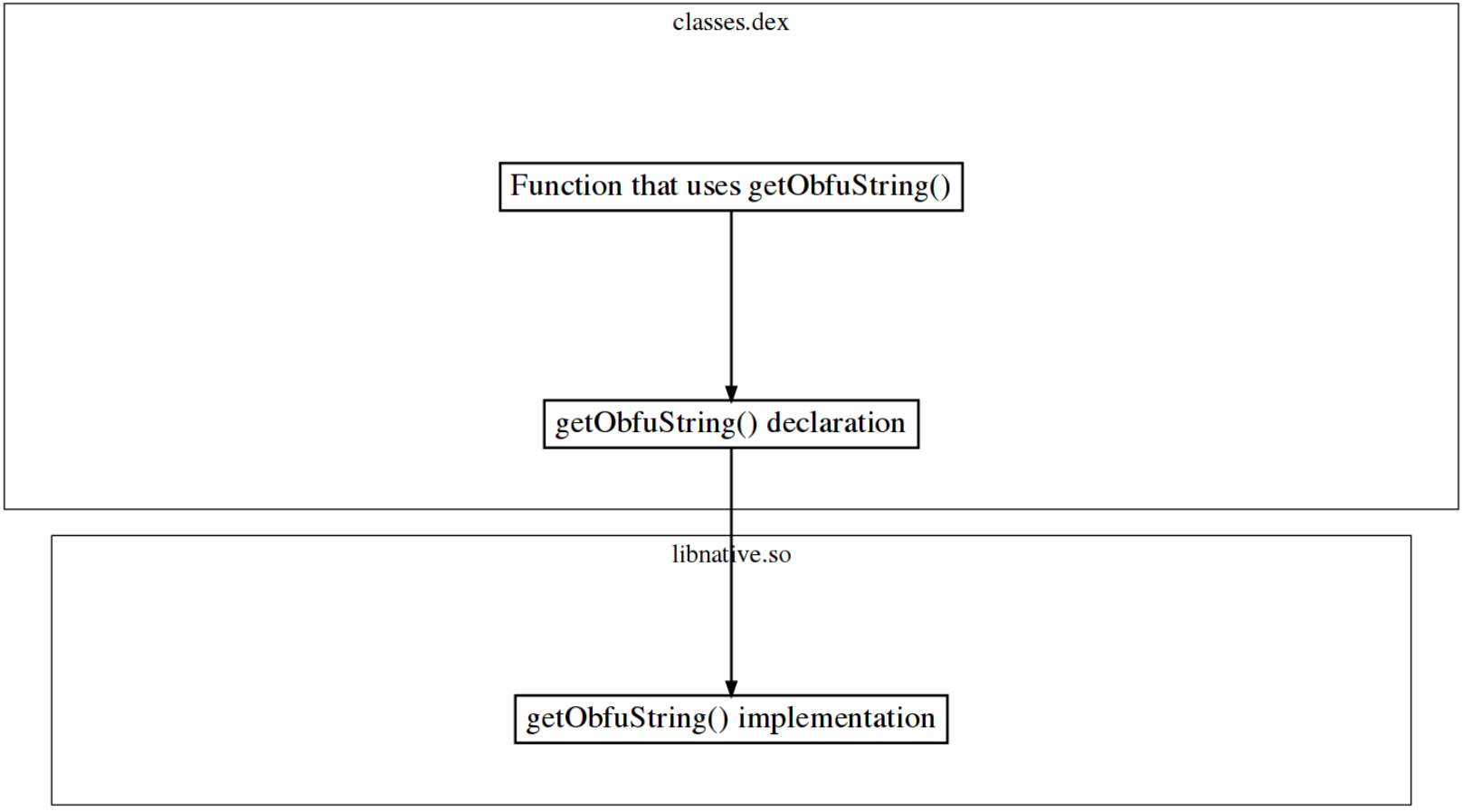

Figure 16:

String literal, Smali

```
# virtual methods
.method method1()V
    .locals 1

    .prologue
    .line 5
    const-string v0, "supersecret"

    .line 6
    return-void
.end method
```

String data removed, Smali

```
# direct methods
.method private static native getObfuString()Ljava/lang/String;
.end method


# virtual methods
.method method1()V
    .locals 1

    .prologue
    .line 5
    invoke-static {}, LClass3;->getObfuString()Ljava/lang/String;
    move-result-object v0

    .line 6
    return-void
.end method
```

String data removed, Diff

```
--- Class3.smali
+++ Class3.smali.obfu
@@ -1,11 +1,17 @@
+# direct methods
+.method private static native getObfuString()Ljava/lang/String;
+.end method
+
+
 # virtual methods
 .method method1()V
     .locals 1

     .prologue
     .line 5
-    const-string v0, "supersecret"
+    invoke-static {}, LClass3;->getObfuString()Ljava/lang/String;
+    move-result-object v0

     .line 6
     return-void
 .end method
```

Function uses string literal

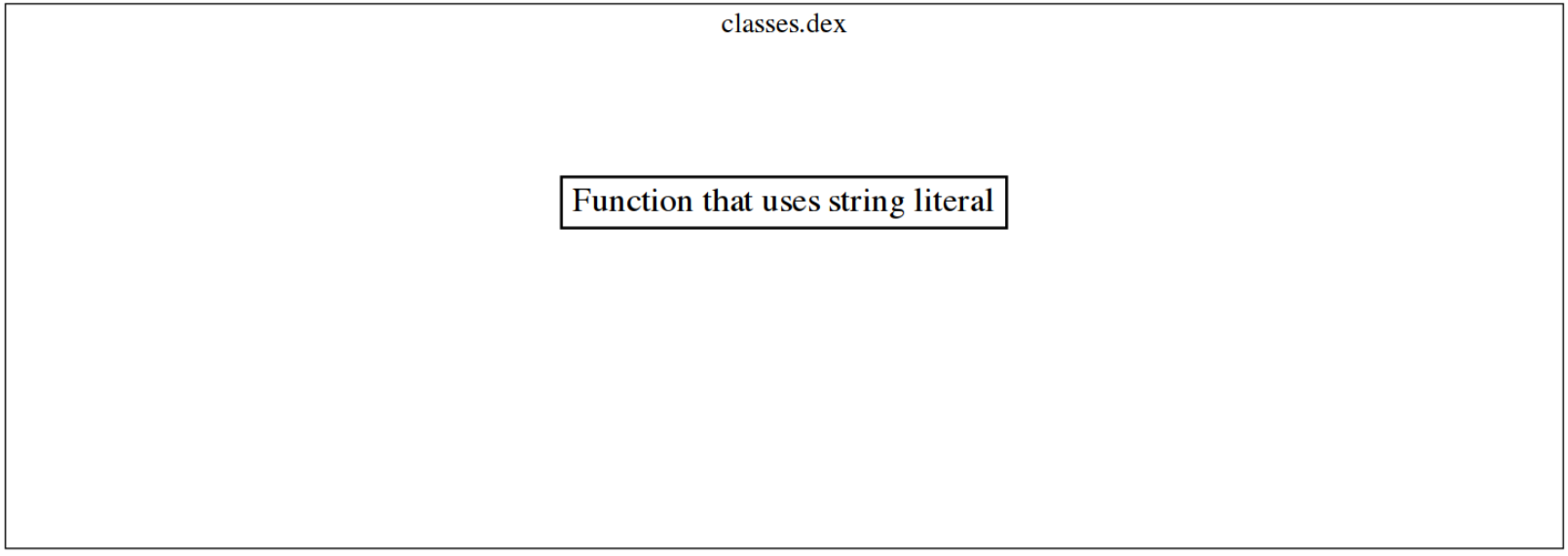

Figure 17:

Function uses getObfuString() with parameter

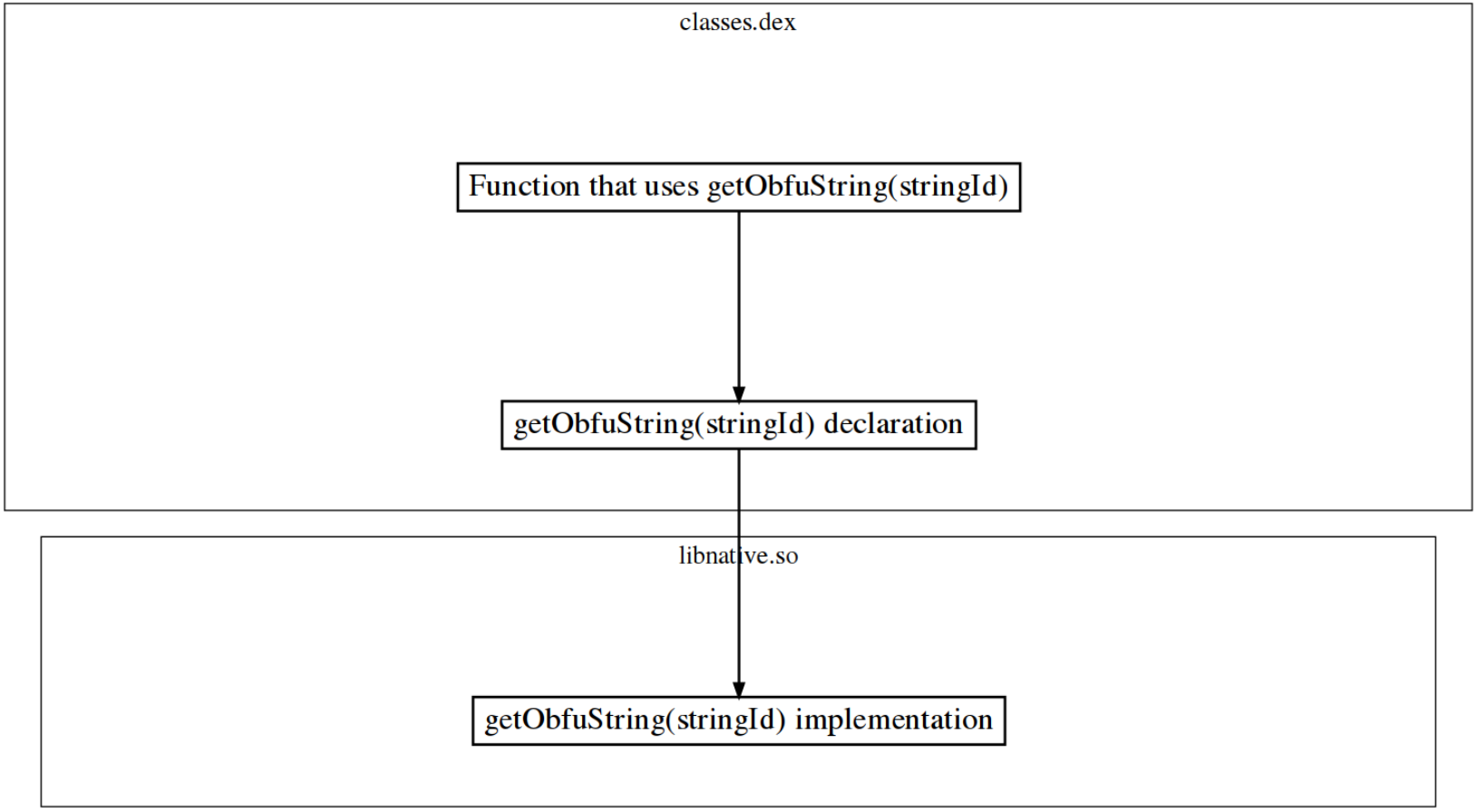

Figure 18:

String literal, Smali

```
# virtual methods
.method method1()V
    .locals 1

    .prologue
    .line 5
    const-string v0, "supersecret"

    .line 6
    return-void
.end method
```

String data removed, Smali

```
# virtual methods
.method method1()V
    .locals 2

    .prologue
    .line 5
    const v1, 0x31337
    invoke-static {v1}, LAnotherClass;->getObfuString(I)Ljava/lang/String;
    move-result-object v0

    .line 6
    return-void
.end method
```

String data removed, Diff

```
--- Class4.smali
+++ Class4.smali.obfu
@@ -1,10 +1,12 @@
 # virtual methods
 .method method1()V
-    .locals 1
+    .locals 2

     .prologue
     .line 5
-    const-string v0, "supersecret"
+    const v1, 0x31337
+    invoke-static {v1}, LAnotherClass;->getObfuString(I)Ljava/lang/String;
+    move-result-object v0

     .line 6
     return-void
```

String data removed

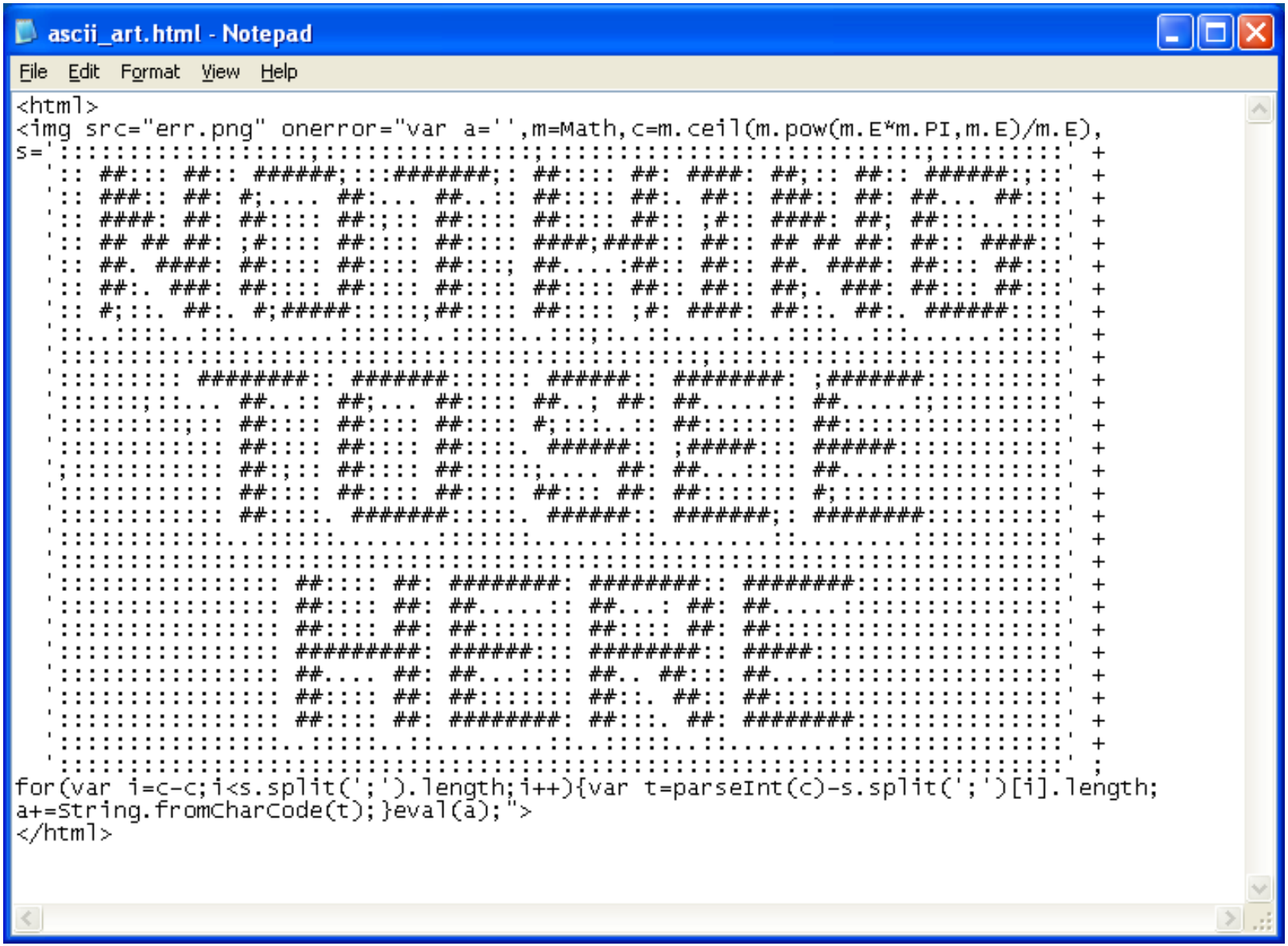

Figure 19:

Some obvious advices

- Choose other names than obfuString and getObfuString()
- Use random char sequences
- Use random valid words from a dictionary
- Use random integers as getObfuString(integerId)
- Try getObfuString(stringId)
- Place fields and functions into other classes

For the previous slide. . .

Figure 20:

Retaliate data extraction

- Combine push-to-field and get-from-function
- Do not export more symbols than needed from the native library
- Use random parameters for getObfuString()
- Crash
- Return fake data
- Call home

Retaliate data extraction

- Check for APK repackaging
- Check integrity of important files yourself
- Retaliate debugging
- Authenticate Java code before native code
  - Password object, constructed in non-obvious way
  - “Function call knocking”
- Encrypt main data with auth data
- Encrypt main data with APK integrity info

Where to store hidden data

- In the native library itself
- In the encrypted files
  - Store only passwords in the library
- Both places

Thanks for attention

Questions?