阿里移动安全

MOBILE SECURITY OF ALIBABA

Android
混淆技巧与反混淆

About Me

小波 Bob Pan	混淆 反混淆
dex2jar	加固 脱壳

pxb1988@gmail.com

Image from: http://www.usmile.at/sites/default/files/publications/201306_obf_report_0.pdf

混淆VS加固

混淆

- 将代码变得难以阅读
- 配置复杂

需要开发配合

加固

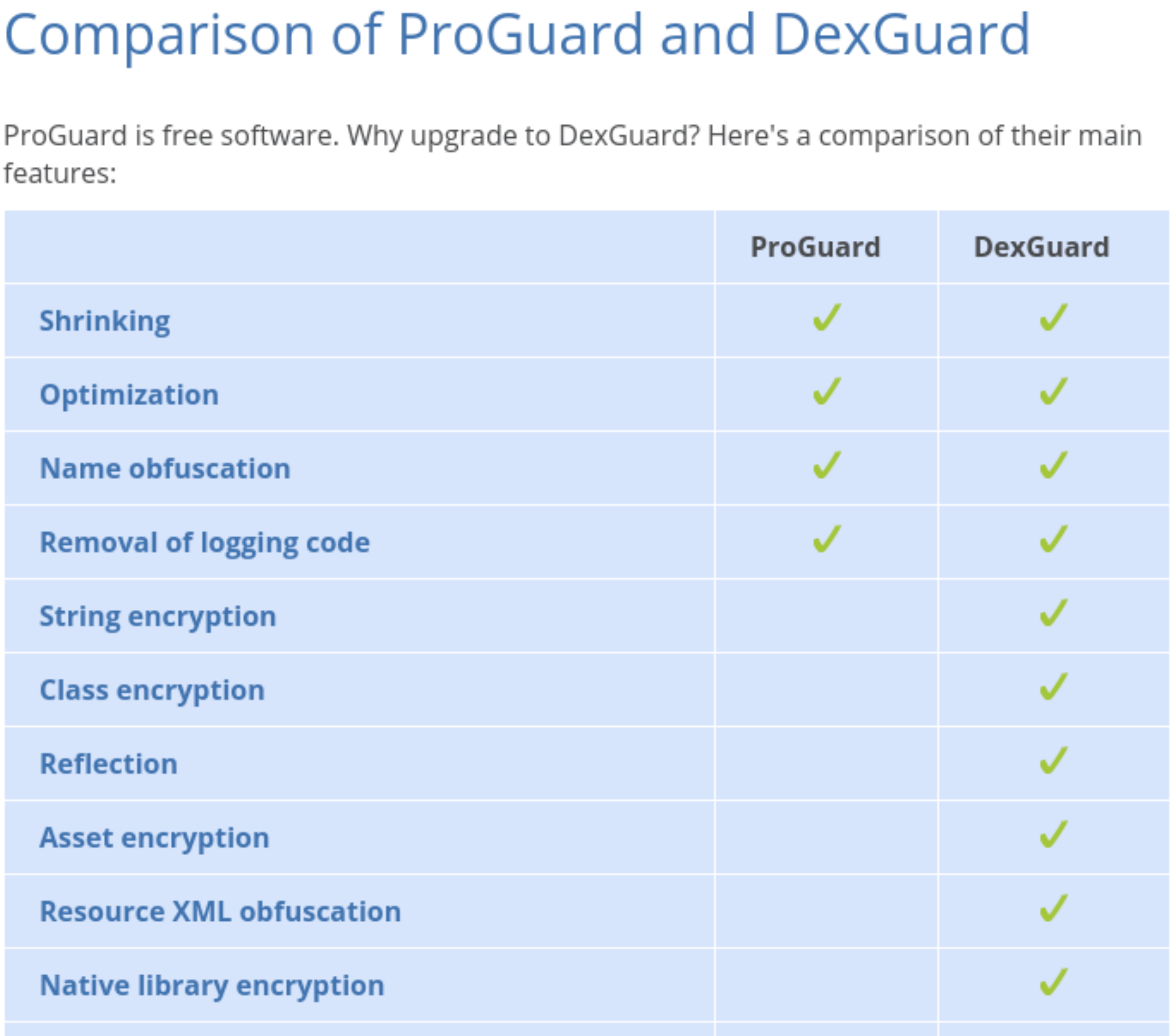
- 隐藏代码
- 对抗自动化工具
- 反调试/反篡改/反注入
- 一键搞定

不需要开发配合

不冲突，可联用！

工具

- ProGuard
- DexGuard



Comparison of ProGuard and DexGuard

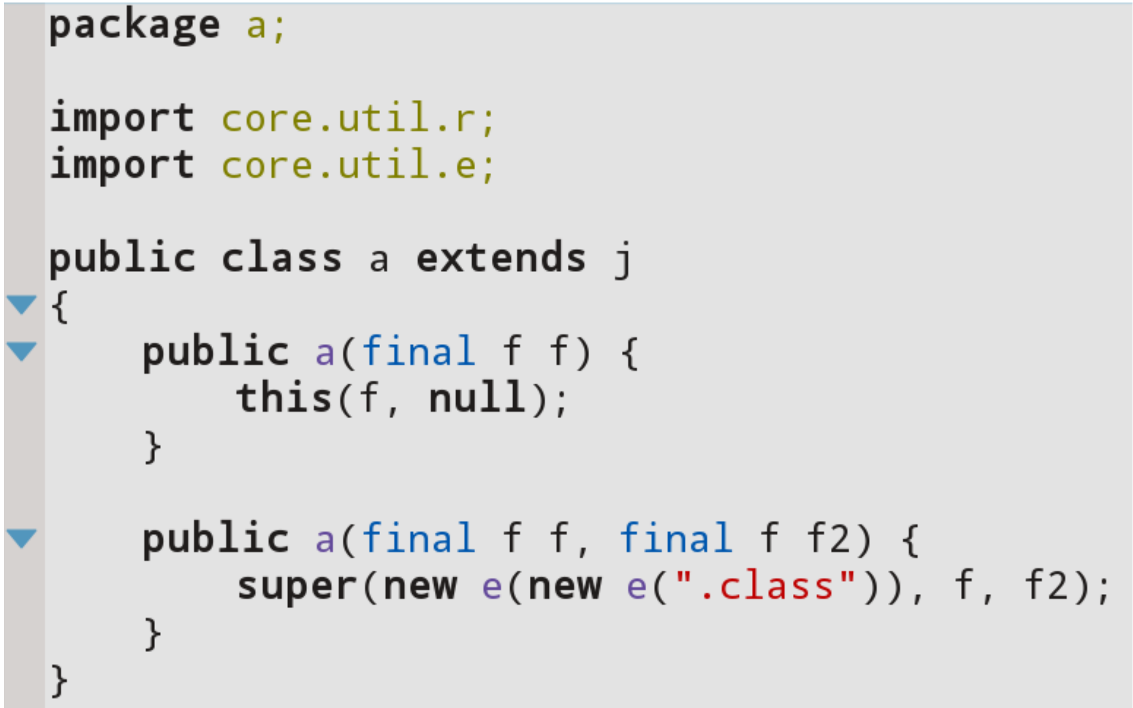
ProGuard is free software. Why upgrade to DexGuard? Here's a comparison of their main features:

	ProGuard	DexGuard
Shrinking	✓	✓
Optimization	✓	✓
Name obfuscation	✓	✓
Removal of logging code	✓	✓
String encryption		✓
Class encryption		✓
Reflection		✓
Asset encryption		✓
Resource XML obfuscation		✓
Native library encryption		✓

Ref: http://www.saikoa.com/comparison-proguard-and-dexguard

名字替换

- 替换类名
- 替换函数名
- 替换成员名
- 替换所有引用



```
package a;

import core.util.r;
import core.util.e;

public class a extends j
{
    public a(final f f) {
        this(f, null);
    }

    public a(final f f, final f f2) {
        super(new e(new e(".class")), f, f2);
    }
}
```

优点:

λ代码可读性差

λ减少文件大小

缺点:

λ接口相关的名字无法替换

λ反射很难自动识别

名字替换:奇葩的名字

- 超长名字

  oooooooooooooo...
- 找茬

  Oo0o0OO00oooOOo0oo

  ijijijjiiiJiIljii
- __$$_$$$$__$$_
- java语法关键字

  int int = 5;

- Unicode
- ȷava \u0237
- CJK字符
- 难以阅读字符

  დამწერლობა

  אָלֶף־בֵּית עִבְרִי
- 盲文点字模型 ⠝

  2800-28FF

名字替换:如何对付奇葩 ？

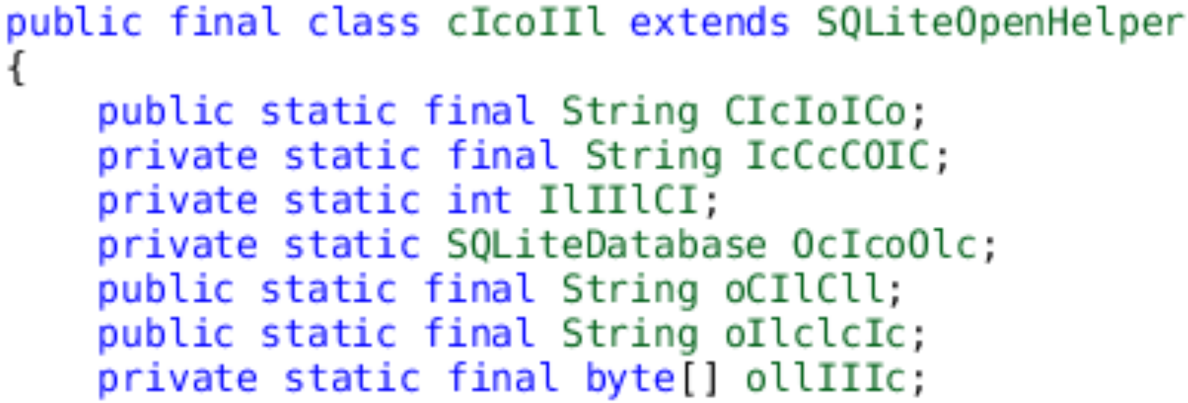
- 相对来说'abc'是比较好阅读的

- Proguard
  再混淆一次!

```
-dontshrink
-dontoptimize
-dontusemixedcaseclassnames
-keepattributes *Annotation*

# 几大组件
-keep public class * extends Activity/Application/...
... #其他keep, 这里略去
-dontwarn **
-printmapping mapping0.txt

-injars obad-dex2jar.jar
-outjars aaa.jar
-libraryjars android.jar
```

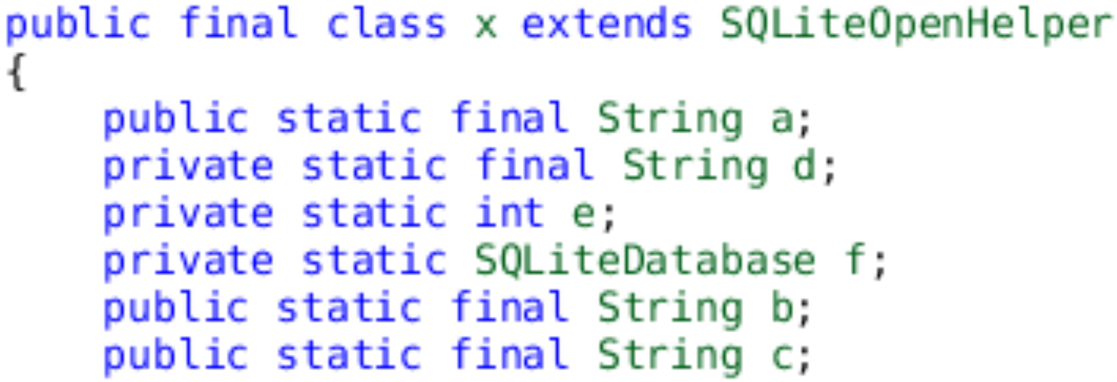
名字替换:结果比较

处理前



```
public final class cIcoIIl extends SQLiteOpenHelper
{
    public static final String CIcIoICo;
    private static final String IcCcCOIC;
    private static int IlIIlCI;
    private static SQLiteDatabase OcIcoOlc;
    public static final String oCIlCll;
    public static final String oIlclcIc;
    private static final byte[] ollIIIc;
```

处理后



```
public final class x extends SQLiteOpenHelper
{
    public static final String a;
    private static final String d;
    private static int e;
    private static SQLiteDatabase f;
    public static final String b;
    public static final String c;
```

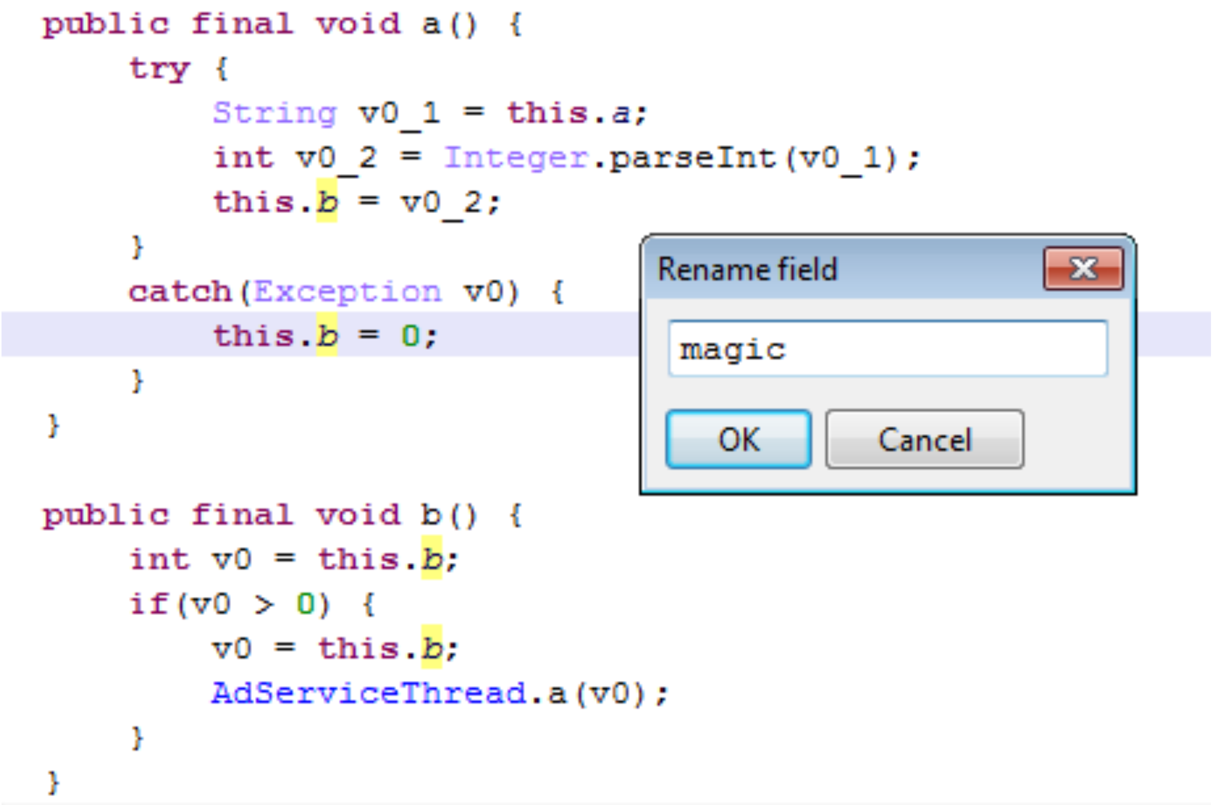
# 名字替换:如何对付abc？

- 没办法自动化,
  只能靠阅读代码
- 高富帅

  JEB

- 普通大众

  Proguard



```
public final void a() {
    try {
        String v0_1 = this.a;
        int v0_2 = Integer.parseInt(v0_1);
        this.b = v0_2;
    }
    catch(Exception v0) {
        this.b = 0;
    }
}

public final void b() {
    int v0 = this.b;
    if(v0 > 0) {
        v0 = this.b;
        AdServiceThread.a(v0);
    }
}
```

Rename field

magic

名字替换: Proguard重命名

λ1. 生成默认的mapping文件

Proguard配置

```
-dontshrink
-dontoptimize
-injars aaa.jar
-libraryjars android.jar
-keep class *
-printmapping mapping1.txt
```

Mapping文件

```
com.android.system.admin.x -> com.android.system.admin.x:
    java.lang.String a -> a
    java.lang.String d -> d
    int e -> e
    ...
```

名字替换: Proguard重命名

λ2. 修改mapping文件, 重新运行Proguard

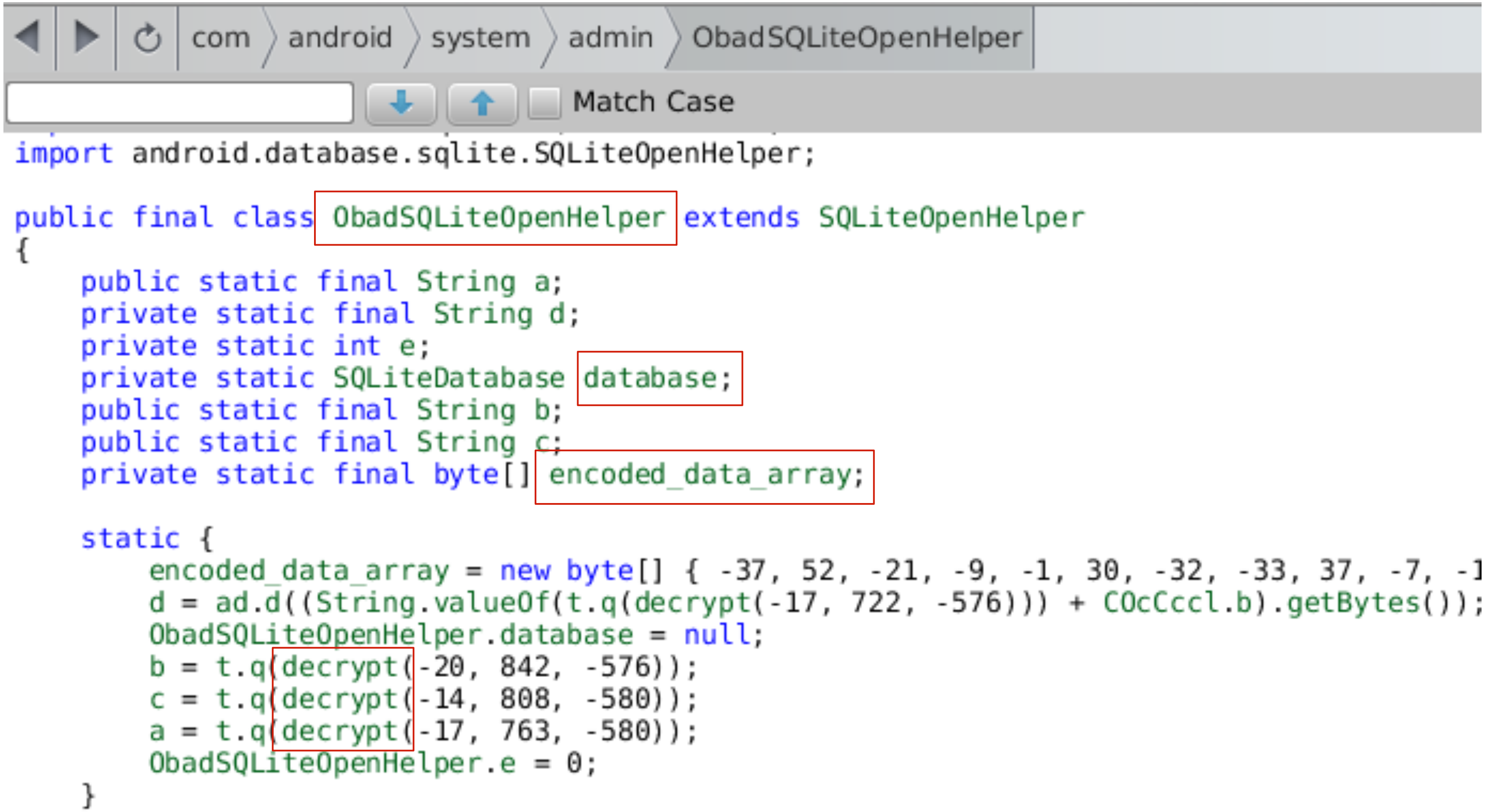
Mapping文件

```
com.android.system.admin.x -> ...ObadSQLiteOpenHelper:
    android.database.sqlite.SQLiteDatabase f -> database
    byte[] g -> encoded_data_array
    java.lang.String a(int,int,int) -> decrypt
```

Proguard配置

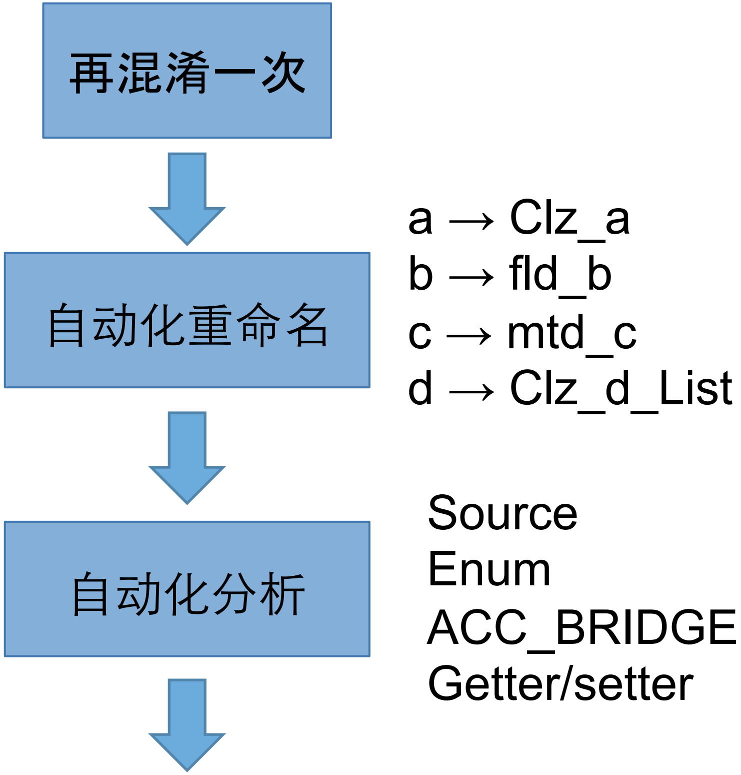
```
-dontshrink
-dontoptimize
-injars aaa.jar
-outjars bbb.jar
-libraryjars android.jar
-applymapping mapping1.txt
```

Proguard重命名结果



```
import android.database.sqlite.SQLiteOpenHelper;

public final class ObadSQLiteOpenHelper extends SQLiteOpenHelper
{
    public static final String a;
    private static final String d;
    private static int e;
    private static SQLiteDatabase database;
    public static final String b;
    public static final String c;
    private static final byte[] encoded_data_array;

    static {
        encoded_data_array = new byte[] { -37, 52, -21, -9, -1, 30, -32, -33, 37, -7, -1
        d = ad.d((String.valueOf(t.q(decrypt(-17, 722, -576))) + COcCccl.b).getBytes());
        ObadSQLiteOpenHelper.database = null;
        b = t.q(decrypt(-20, 842, -576));
        c = t.q(decrypt(-14, 808, -580));
        a = t.q(decrypt(-17, 763, -580));
        ObadSQLiteOpenHelper.e = 0;
    }
```

反混淆大项目(名字恢复)



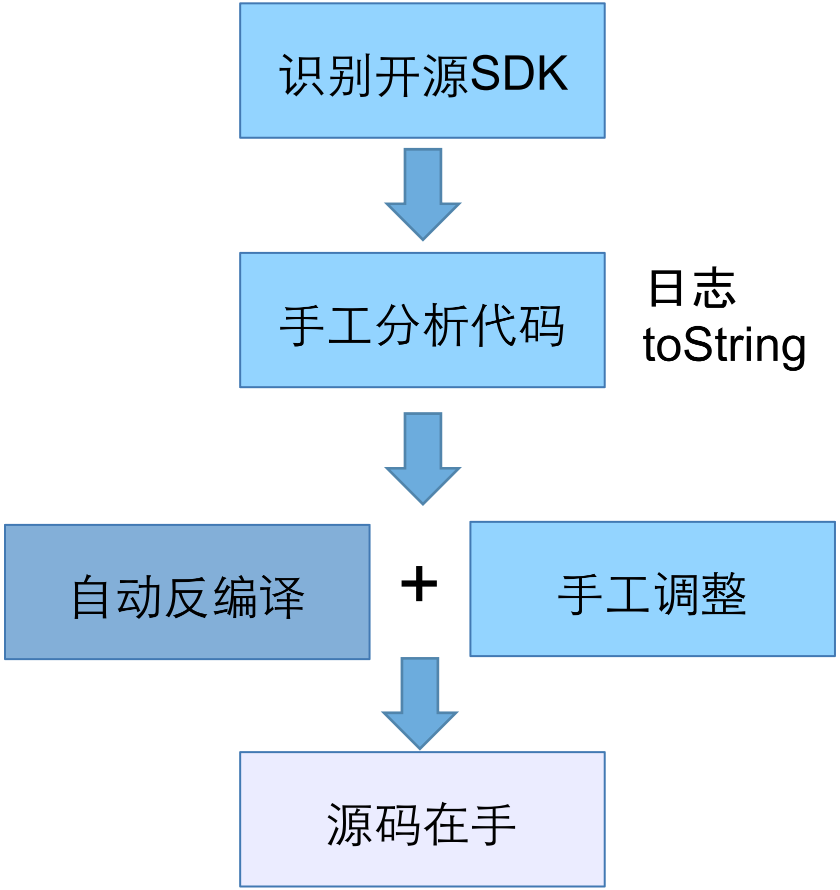
再混淆一次

自动化重命名

a → Clz_a
b → fld_b
c → mtd_c
d → Clz_d_List

自动化分析

Source
Enum
ACC_BRIDGE
Getter/setter



识别开源SDK

手工分析代码

日志
toString

自动反编译

+

手工调整

源码在手

字符串加密

- 将字符串在运行时恢复
- DexGuard
  - String a(int, int, int)
- Other
  - String a(String)

```
Class.forName(a(130, 1, -10))
.getMethod(a(53, 19, -21),
 Class.forName(a(79, 1, -11)))
.invoke(j, instance);
```

优点:

λ静态看不到字符串

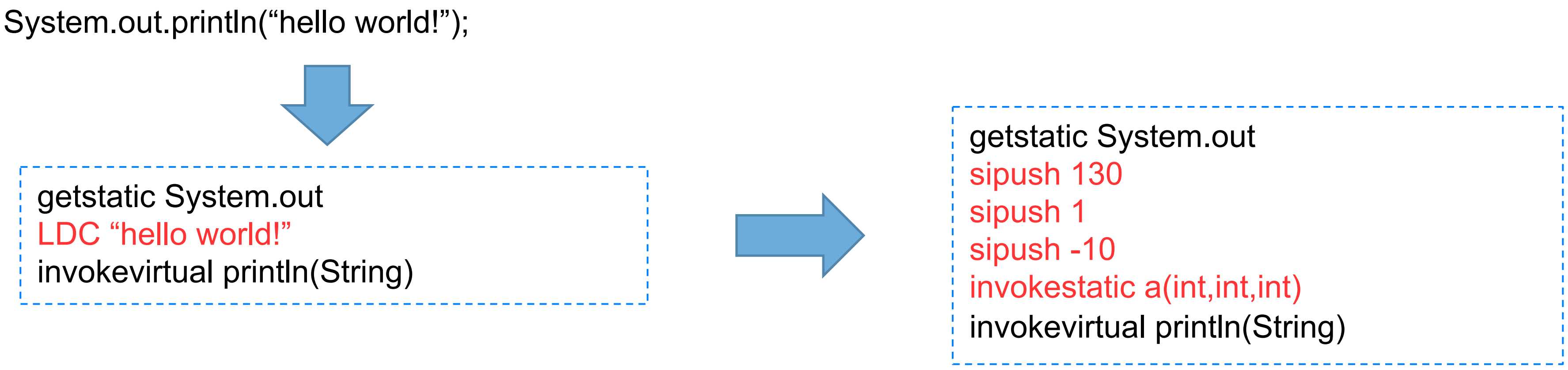
缺点:

λ内存消耗增加

λ性能降低

字符串加密:简单实现

- Java bytecode 使用LDC指令加载字符串
- 替换对应的LDC指令即可实现加密



```
System.out.println(“hello world!”);
```

```
getstatic System.out
LDC “hello world!”
invokevirtual println(String)
```

```
getstatic System.out
sipush 130
sipush 1
sipush -10
invokestatic a(int,int,int)
invokevirtual println(String)
```

字符串加密:带来的问题

λ使用'==' 比较字符串

```
void fa(){
  fb(“1.0”);
}
void fb(String version) {
  if(version == “1.0”){
    print(“yes!”);
  }
}
```

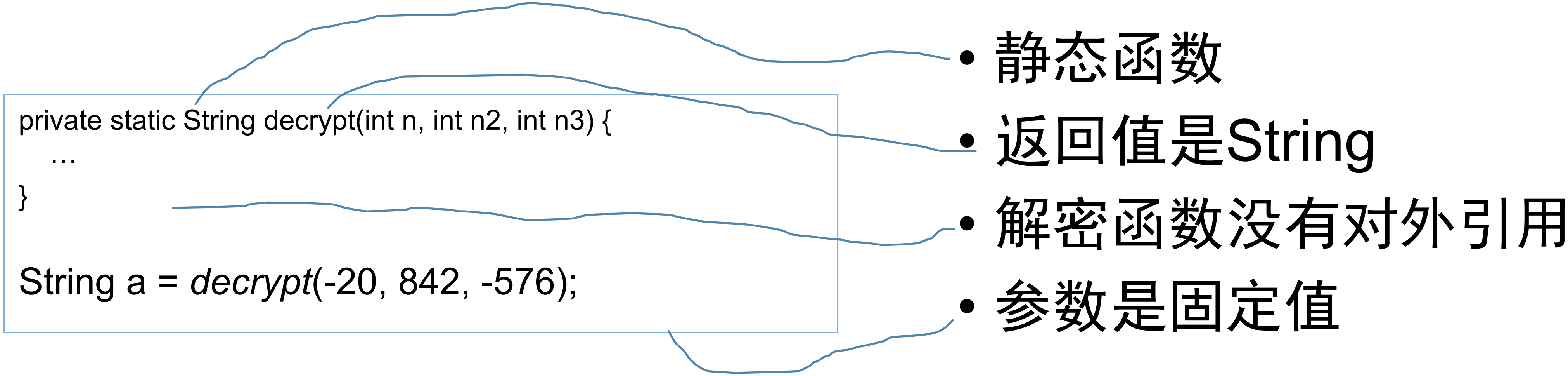
条件成立, 打印yes

```
void fa(){
  fb(new String(...));
}
void fb(String version) {
  if(version == new String(...)){
    print(“yes!”);
  }
}
```

条件不成立, 什么都没有

λ解决办法: 使用'equals' 比较字符串

字符串加密: 如何应对?



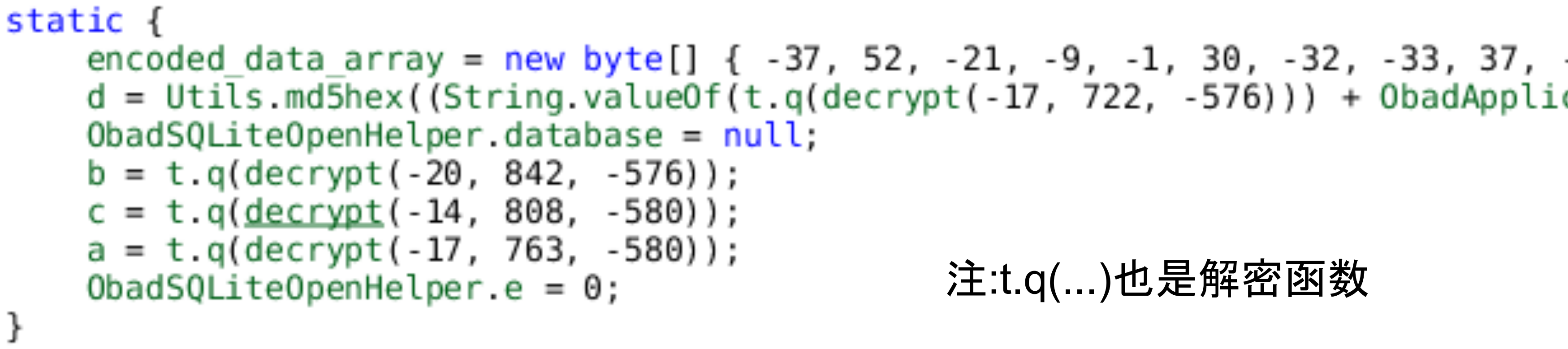
```
private static String decrypt(int n, int n2, int n3) {
    ...
}

String a = decrypt(-20, 842, -576);
```

- 静态函数
- 返回值是String
- 解密函数没有对外引用
- 参数是固定值

解决办法:
找到对应的函数和参数, 反射调用, 将结果写回.

字符串解密结果

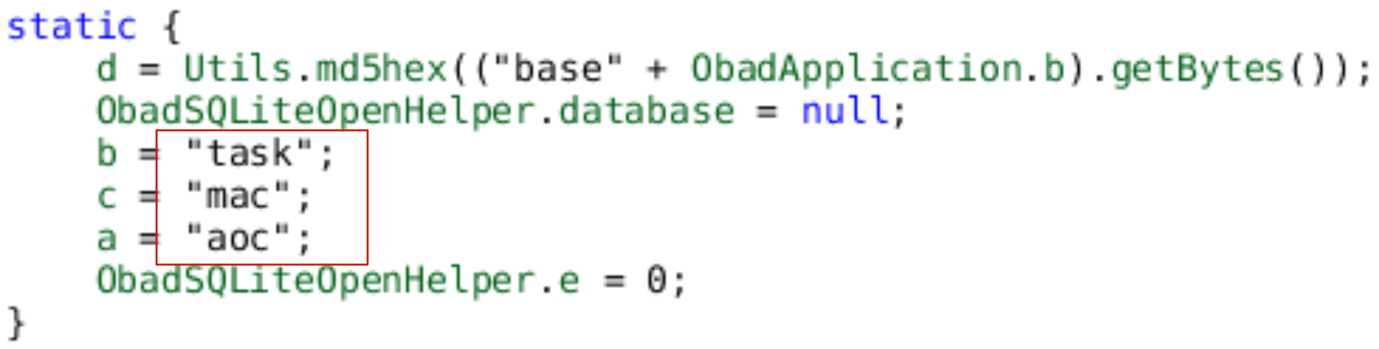
解密前



```
static {
    encoded_data_array = new byte[] { -37, 52, -21, -9, -1, 30, -32, -33, 37,
    d = Utils.md5hex((String.valueOf(t.q(decrypt(-17, 722, -576))) + ObadApplic
    ObadSQLiteOpenHelper.database = null;
    b = t.q(decrypt(-20, 842, -576));
    c = t.q(decrypt(-14, 808, -580));
    a = t.q(decrypt(-17, 763, -580));
    ObadSQLiteOpenHelper.e = 0;
}
```

注:t.q(...)也是解密函数

解密后



```
static {
    d = Utils.md5hex(("base" + ObadApplication.b).getBytes());
    ObadSQLiteOpenHelper.database = null;
    b = "task";
    c = "mac";
    a = "aoc";
    ObadSQLiteOpenHelper.e = 0;
}
```

# 反射替换

- 将函数替换为等价的反射API调用

```
String c = "abc".substring(2,3);
```

```
String c = (String)Class.forName("java.lang.String")
        .getMethod("substring", int.class, int.class)
        .invoke("abc", 2, 3)
```

优点:

λ与字符加密串结合效果更佳

缺点:

λ代码大小增加

λ**性能降低**

反射替换: 简单实现

```
String c =
    "abc".substring(2,3);
```

Local	Stack	Opcode
		ldc "abc"
	“abc”	sipush 2
	“abc”, 2	sipush 3
	“abc”, 2, 3	invokevirtual substring(II)

思路:

1.将Stack的数据保存到Local
2.构建Class对象
3.构建Method对象
4.重新加载Local中的值到Stack
5.调用invoke函数

反射替换: 简单实现

Local	Stack	Opcode
	“abc”, 2, 3	astore 1, istore 2, istore 3
“abc”, 2, 3		ldc “java.lang.String” invokestatic Class.forName
“abc”, 2, 3	String.class	ldc “substring”, ... #构建参数类型 invokevirtual Class.getMethod
“abc”, 2, 3	substring	aload 1, iload 2, iload 3,
“abc”, 2, 3	substring, “abc”, [2, 3]	invokevirtual Method.invoke()

等价于

```
String a=”abc”; int b=2; int c=3;
String c = (String)Class.forName("java.lang.String")
        .getMethod("substring", int.class, int.class)
        .invoke(a, b, c)
```

反射替换:如何处理?

- 1. 将所有的Class.forName恢复成class对象

```
String c = (String)Class.forName("java.lang.String")
        .getMethod("substring", int.class, int.class)
        .invoke("abc", 2, 3)
```

```
String c = (String)String.class
        .getMethod("substring", int.class, int.class)
        .invoke("abc", 2, 3)
```

# 反射替换:如何处理?

- 2. 将getMethod恢复成对应对象

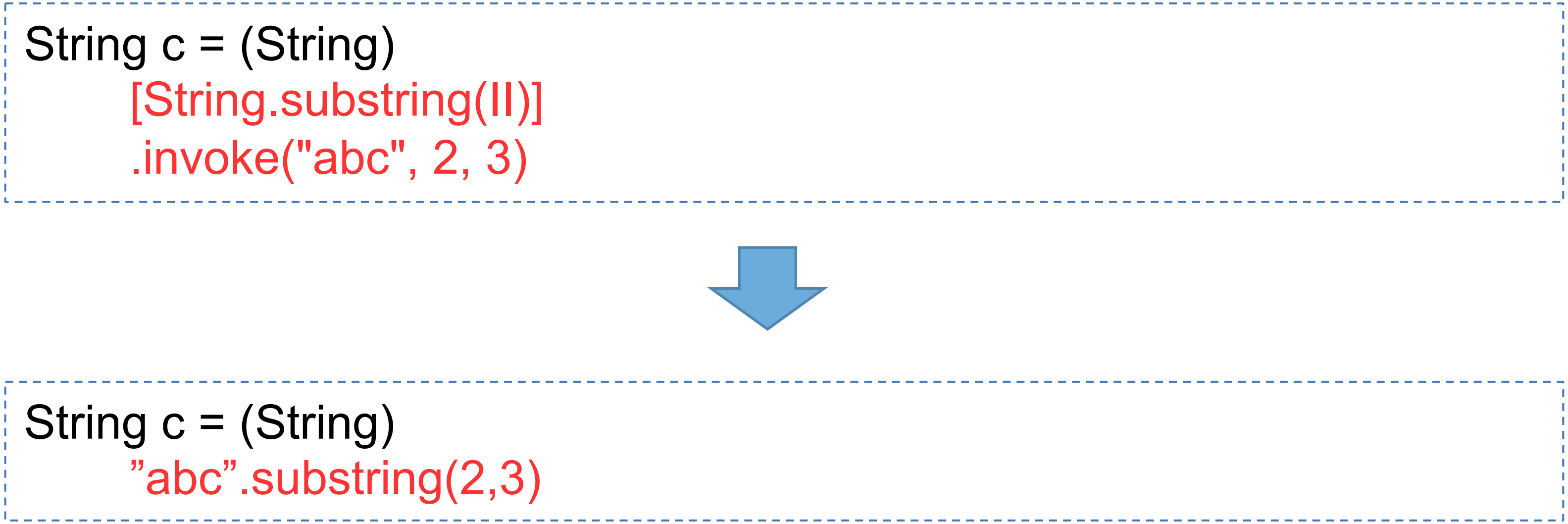
```
String c = (String)String.class
        .getMethod("substring", int.class, int.class)
        .invoke("abc", 2, 3)
```

```
String c = (String)
        [String.substring(II)]
        .invoke("abc", 2, 3)
```

表示一个Method对象

反射替换:如何处理?

- 3. 将invoke函数展开

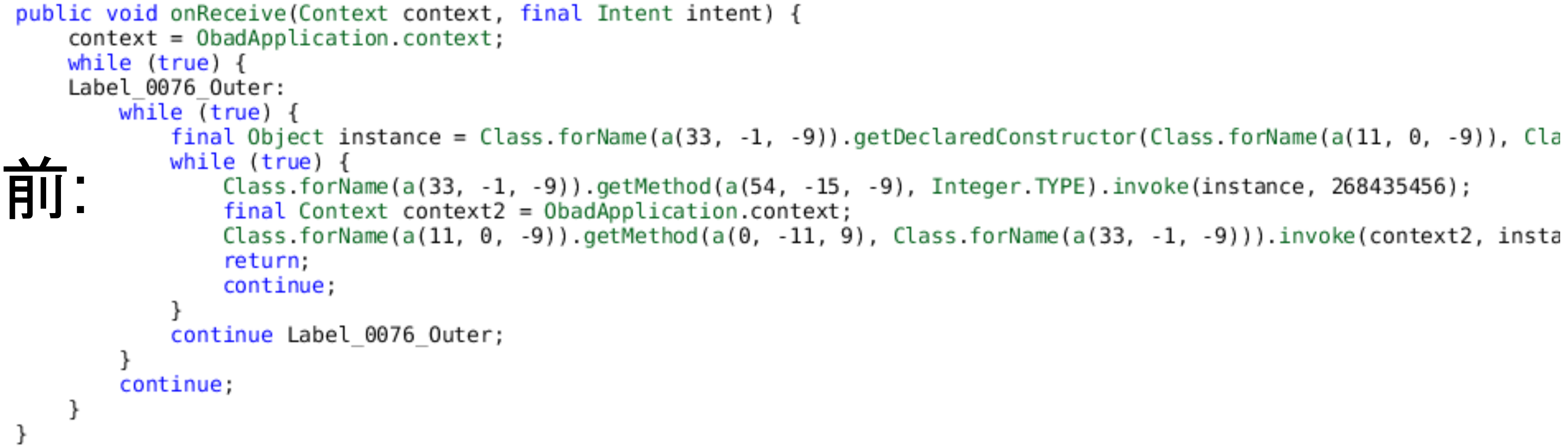


```
String c = (String)
        [String.substring(II)]
        .invoke("abc", 2, 3)
```

```
String c = (String)
        ”abc”.substring(2,3)
```

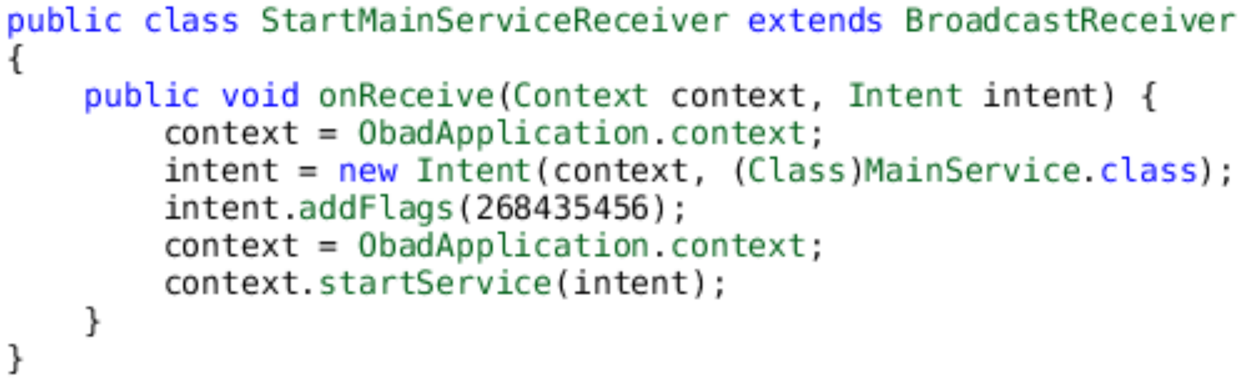
清理反射结果

清理前:



```
public void onReceive(Context context, final Intent intent) {
    context = ObadApplication.context;
    while (true) {
    Label_0076_Outer:
        while (true) {
            final Object instance = Class.forName(a(33, -1, -9)).getDeclaredConstructor(Class.forName(a(11, 0, -9)), Cla
            while (true) {
                Class.forName(a(33, -1, -9)).getMethod(a(54, -15, -9), Integer.TYPE).invoke(instance, 268435456);
                final Context context2 = ObadApplication.context;
                Class.forName(a(11, 0, -9)).getMethod(a(0, -11, 9), Class.forName(a(33, -1, -9))).invoke(context2, insta
                return;
                continue;
            }
            continue Label_0076_Outer;
        }
        continue;
    }
}
```

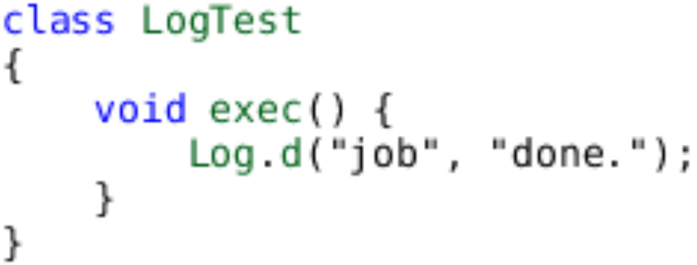
清理后:



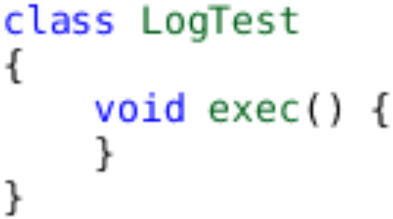
```
public class StartMainServiceReceiver extends BroadcastReceiver
{
    public void onReceive(Context context, Intent intent) {
        context = ObadApplication.context;
        intent = new Intent(context, (Class)MainService.class);
        intent.addFlags(268435456);
        context = ObadApplication.context;
        context.startService(intent);
    }
}
```

# 日志清除

- 清理android日志输出代码



```
class LogTest
{
    void exec() {
        Log.d("job", "done.");
    }
}
```



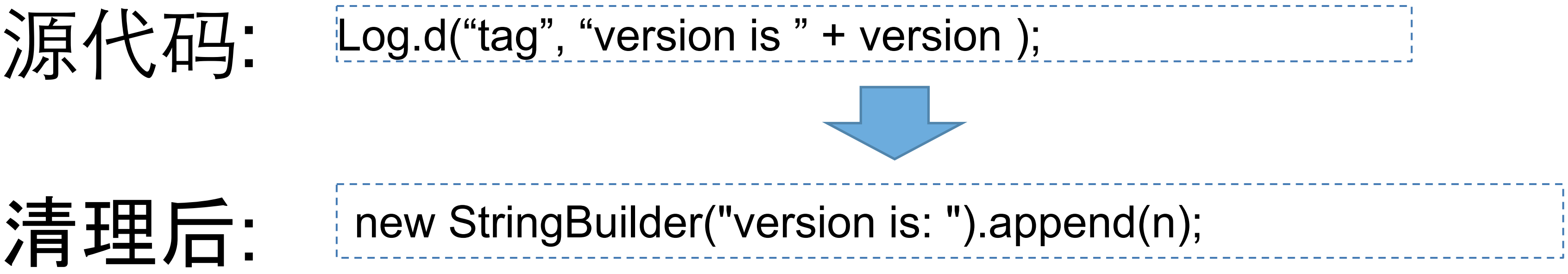
```
class LogTest
{
    void exec() {
    }
}
```

- 实现原理

```
-assumenosideeffects class android.util.Log {
    public static *** d(...);
    public static *** w(...);
    public static *** v(...);
    public static *** i(...);
}
```

Ref: http://stackoverflow.com/questions/5553146/disable-logcat-output-completely-in-release-android-app/5553290#5553290

日志清除: Proguard缺陷



源代码:

```
Log.d(“tag”, “version is ” + version );
```

清理后:

```
new StringBuilder("version is: ").append(n);
```

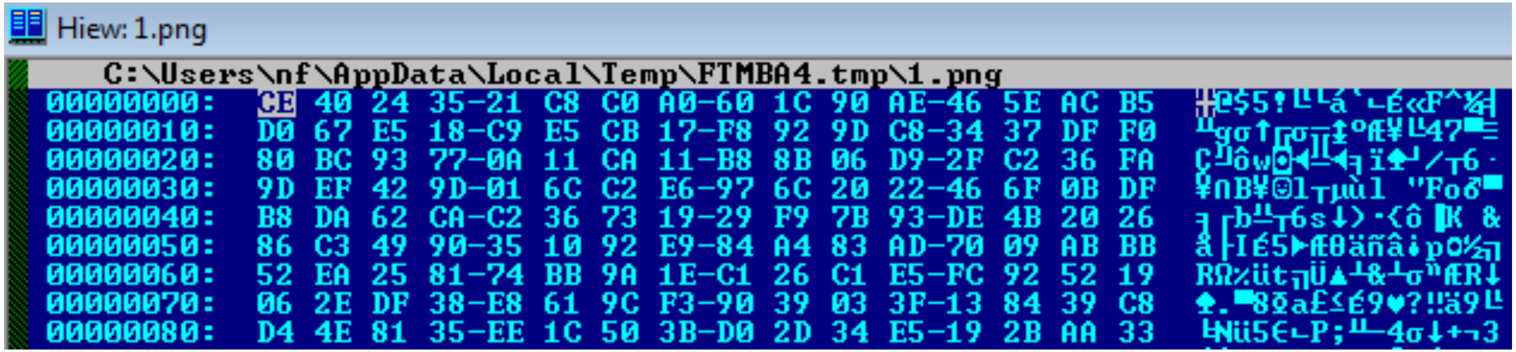
原因:

“version is” + version 会被转换成
new StringBuilder(“version is ”).append(version).toString().
而Proguard只负责删除Log.d的函数调用, 没有删除StringBuilder相关的代码

Ref: http://stackoverflow.com/questions/22713166/removing-log-calls-with-proguard-leaves-behind-stringbuilders

Asset加密

- 将apk中asset目录的文件加密，使用前解密



```
   C:\Users\nf\AppData\Local\Temp\FTMBA4.tmp\1.png
00000000:  CE 40 24 35-21 C8 C0 A0-60 1C 90 AE-46 5E AC B5  ╬@$5!╚└á`∟É«F^¼╡
00000010:  D0 67 E5 18-C9 E5 CB 17-F8 92 9D C8-34 37 DF F0  ╨gσ↑╔σ╦↨°Æ¥╚47▀≡
00000020:  80 BC 93 77-0A 11 CA 11-B8 8B 06 D9-2F C2 36 FA  Ç╝ôw◙◄╩◄╕ï♠┘/┬6·
00000030:  9D EF 42 9D-01 6C C2 E6-97 6C 20 22-46 6F 0B DF  ¥∩B¥☺l┬µùl "Fo♂▀
00000040:  B8 DA 62 CA-C2 36 73 19-29 F9 7B 93-DE 4B 20 26  ╕┌b╩┬6s↓)∙{ô▐K &
00000050:  86 C3 49 90-35 10 92 E9-84 A4 83 AD-70 09 AB BB  å├IÉ5►ÆΘäñâ¡p○½╗
00000060:  52 EA 25 81-74 BB 9A 1E-C1 26 C1 E5-FC 92 52 19  RΩ%üt╗Ü▲┴&┴σⁿÆR↓
00000070:  06 2E DF 38-E8 61 9C F3-90 39 03 3F-13 84 39 C8  ♠.▀8Φa£≤É9♥?‼ä9╚
00000080:  D4 4E 81 35-EE 1C 50 3B-D0 2D 34 E5-19 2B AA 33  ╘Nü5ε∟P;╨-4σ↓+¬3
```

优点:

λ隐藏资源于无形

缺点:

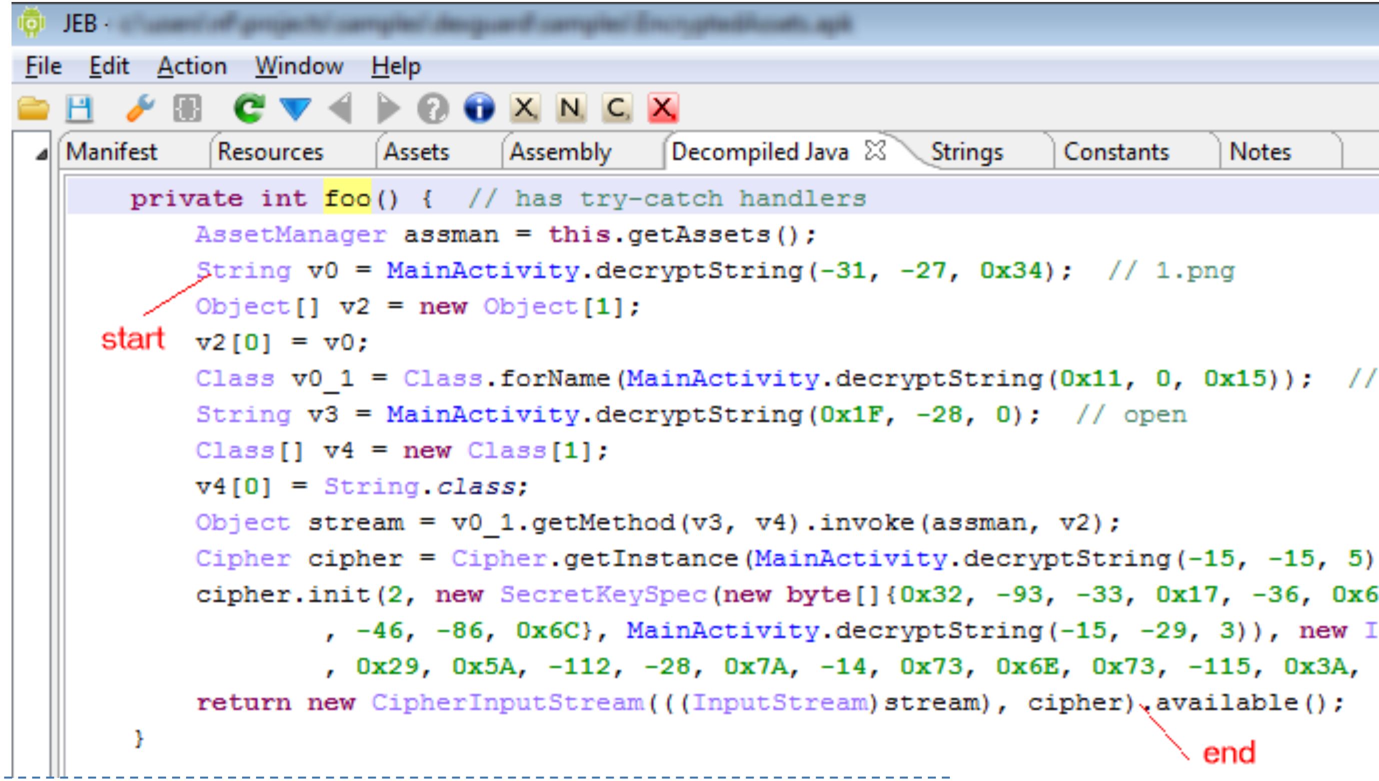
λ必须调用AssetManager.open()

λ性能降低

Ref: http://www.android-decompiler.com/blog/2013/04/07/dexguards-assets-encryption/

Asset加密: 原理

- 拦截open函数返回解密流



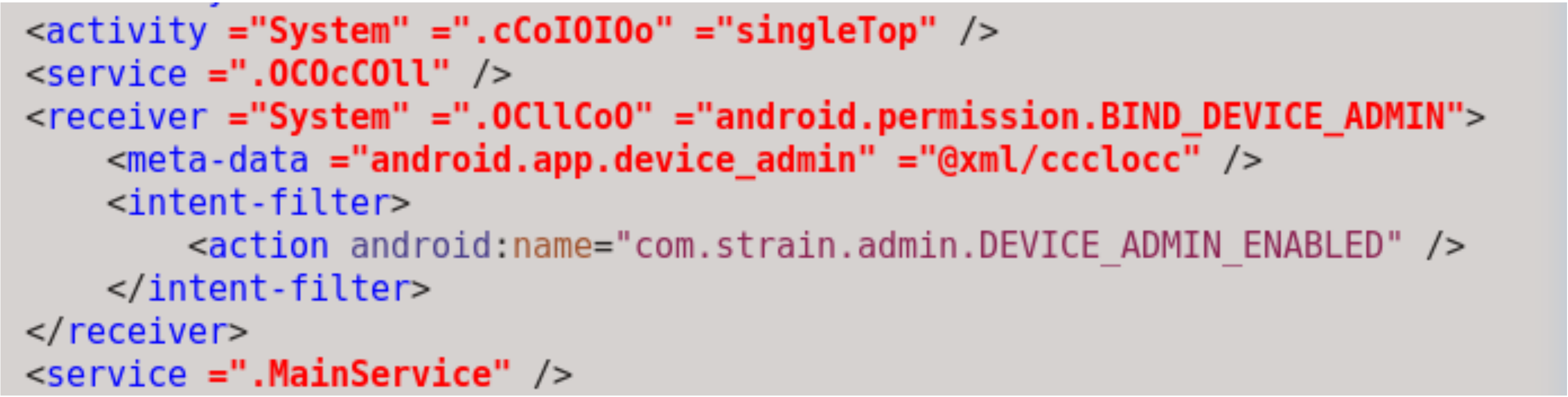
```
    private int foo() {  // has try-catch handlers
        AssetManager assman = this.getAssets();
        String v0 = MainActivity.decryptString(-31, -27, 0x34);  // 1.png
        Object[] v2 = new Object[1];
        v2[0] = v0;
        Class v0_1 = Class.forName(MainActivity.decryptString(0x11, 0, 0x15));  //
        String v3 = MainActivity.decryptString(0x1F, -28, 0);  // open
        Class[] v4 = new Class[1];
        v4[0] = String.class;
        Object stream = v0_1.getMethod(v3, v4).invoke(assman, v2);
        Cipher cipher = Cipher.getInstance(MainActivity.decryptString(-15, -15, 5)
        cipher.init(2, new SecretKeySpec(new byte[]{0x32, -93, -33, 0x17, -36, 0x6
                , -46, -86, 0x6C}, MainActivity.decryptString(-15, -29, 3)), new I
                , 0x29, 0x5A, -112, -28, 0x7A, -14, 0x73, 0x6E, 0x73, -115, 0x3A,
        return new CipherInputStream(((InputStream)stream), cipher).available();
    }
```

等价于

```
InputStream is = this.getAssets().open();
Cipher cipher = Cipher.getInstance("AES/CFB/NoPadding");
cipher.init(DECRYPT_MODE, /* key */);
return new CipherInputStream(is, cipher).available();
```

AndroidManifest混淆

- namespace和name信息被清除



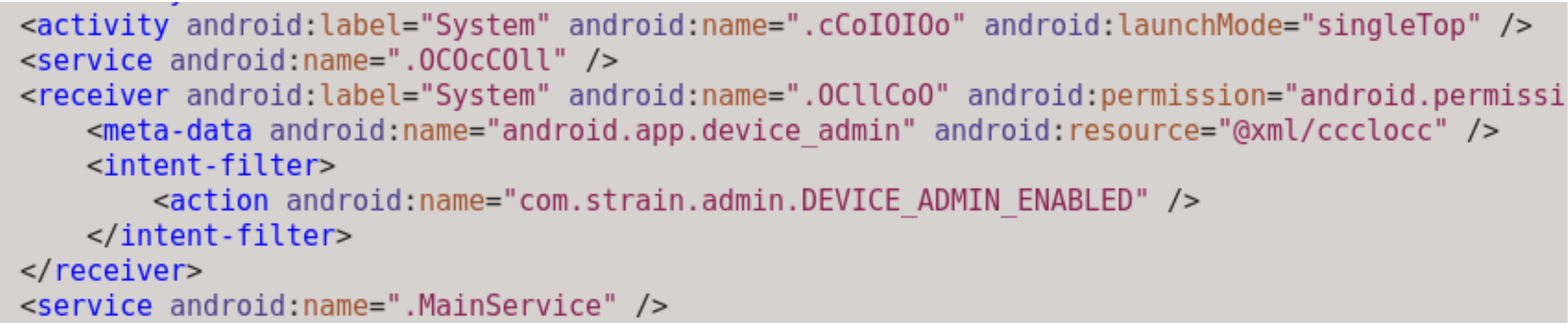
```
<activity ="System" =".cCoIOIOo" ="singleTop" />
<service =".OCOcCOll" />
<receiver ="System" =".OCllCoO" ="android.permission.BIND_DEVICE_ADMIN">
    <meta-data ="android.app.device_admin" ="@xml/ccclocc" />
    <intent-filter>
        <action android:name="com.strain.admin.DEVICE_ADMIN_ENABLED" />
    </intent-filter>
</receiver>
<service =".MainService" />
```

原理:

AndroidManifest中同时包含ResourceId和namespace/name信息.
而Android部分使用ResourceId查找对应的xml标签.
这部分的namespace/name信息是多余的, 可以删除.

AndroidManifest恢复

- 根据ResourceId恢复namesapce/name
- Apktool已经支持读取
- axml工具也可以



```
<activity android:label="System" android:name=".cCoIOIOo" android:launchMode="singleTop" />
<service android:name=".OCOcCOll" />
<receiver android:label="System" android:name=".OCllCoO" android:permission="android.permissi
    <meta-data android:name="android.app.device_admin" android:resource="@xml/ccclocc" />
    <intent-filter>
        <action android:name="com.strain.admin.DEVICE_ADMIN_ENABLED" />
    </intent-filter>
</receiver>
<service android:name=".MainService" />
```

小结

混淆项	恢复	是否可自动化	难度
名字替换	人工恢复	X	999999999
日志清除	X	X	X
字符串加密	静态分析+动态运行	可以	5
反射替换	静态分析	可以	4
Assert加密	可以恢复	半自动化	2
XML混淆	可以恢复	现成工具	1

# 混淆建议

- 减少-keep的数量
  - SDK
  - JNI代码
  - 反射
  - 序列化/反序列化
- 清除其他线索
  - 清除SourceFile
  - 避免日志输出
- 混用混淆项
- 写烂代码
- 加固jaq.taobao.com

Q & A

pxb1988@gmail.com