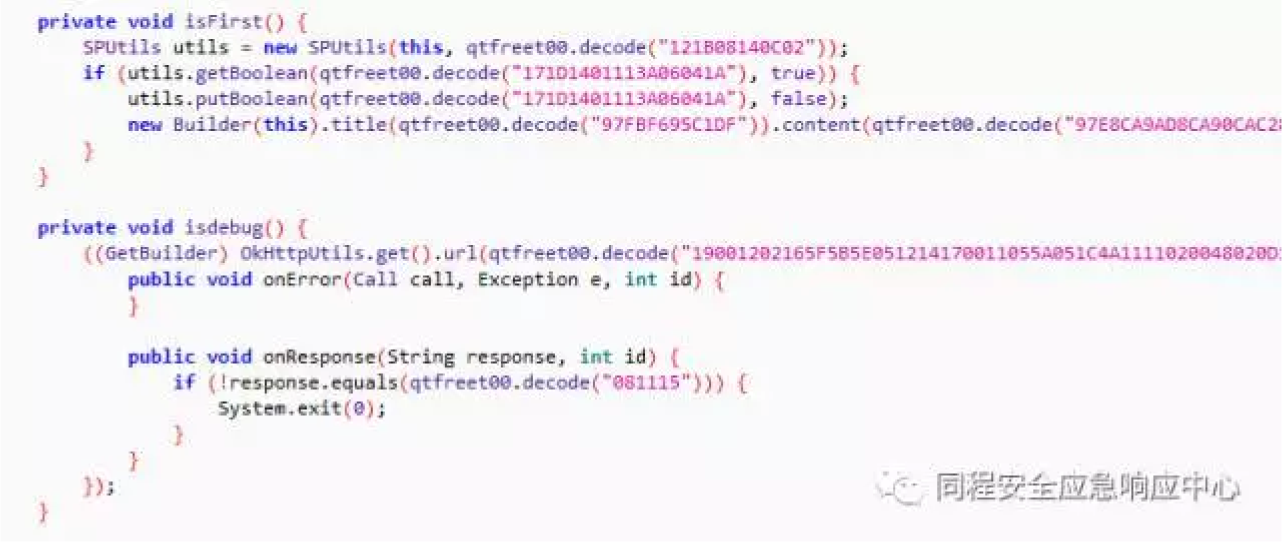
# Android string and dictionary confusion open source implementation

Original 2017-03-06 YSRC Tong Cheng Security Emergency Response Center

Confusing App on **Android** is a common thing, the goal is to protect the app's logic in a certain program is not reversed, but also can reduce the app's size.

The commonly used obfuscation tool in development must first mention ProGuard. This is a free tool. By default, it has been integrated by the Android IDE. The developer only needs to properly configure the obfuscation rules to use proguard to confuse the app, but since it is free The limitations of software are naturally large. The first is that string confusion is not supported. If you want to use string confusion, you need to use its commercial version of DexGuard.

Since there is no free string obfuscation tool, then first implement one, analyze the JEB (Android decompilation tool), Zelix (JAVA obfuscator), BurpSuite (network proxy tool) and other confounding effect of better software, understand After they realized their ideas, they wrote a simple string obfuscation tool, first on the effect:



```
private void isFirst() {
    SPUtils utils = new SPUtils(this, qtfreet00.decode("121B08140C02"));
    if (utils.getBoolean(qtfreet00.decode("171D1401113A06041A"), true)) {
        utils.putBoolean(qtfreet00.decode("171D1401113A06041A"), false);
        new Builder(this).title(qtfreet00.decode("97FBF695C1DF")).content(qtfreet00.decode("97E8CA9AD8CA90CAC2
    }
}

private void isdebug() {
    ((GetBuilder) OkHttpUtils.get().url(qtfreet00.decode("19001202165F5B5E05121417001105 5A051C4A1111020048020D
        public void onError(Call call, Exception e, int id) {
        }

        public void onResponse(String response, int id) {
            if (!response.equals(qtfreet00.decode("081115"))) {
                System.exit(0);
            }
        }
    });
}
```

It can be seen that the string here has been processed into hexadecimal, and each time it is executed, it will call the decode method to restore the string. The decode method is also very simple.

```
static public String decode ( String STR) {
    ByteArrayOutputStream BAOS = new new ByteArrayOutputStream (str.length () / 2 );
    // Each 2 bit 16 hexadecimal integer assembled into a byte
for ( int I = 0 ; I < STR. Length (); i += 2 ) baos .write(( hexString .indexOf(str.charAt( i )) << 4 | hexStrin
        1 ))));
    byte [] b = baos .toByteArray();
    int len = b . length ;
    int keyLen = KEY .length();
    for ( int i = 0 ; i < len ; i ++ ) {
        b [ i ] = ( byte ) ( b [ i ] ^ KEY .charAt( i % keyLen ));
    }
    return new String( b );
}
```

The principle is very simple, that is how to achieve it, the implementation of ideas here is to deal with in the smali layer, that is, in the App compiler to generate apk after processing, use apktool to apk decompile, and then confuse the smali string .

Since it is confusing, it is necessary to iterate over each smali file, which is very simple

```
Private static void getFiles ( String filePath) {
    File [] files = new File(filePath).listFiles();
    if ( files == null ) {
        return ;
    }
    for ( File file : files ) {
        if ( file .isDirectory() ) {
            getFiles ( File .getPath ());
        } the else {
            Filelist .add ( File .getPath ());
        }
    }
}
```

Use iterative mode to add each file to the list. After that, it will process the traversed files directly. Codes will be written directly and the comments will be written clearly.

```
Private static void FileTofindString ( String path) {
    StringBuilder sb = new StringBuilder();
    try {
        InputStreamReader read = new InputStreamReader( new FileInputStream(path), "UTF-8" );
        BufferedReader br = new BufferedReader( read );
        String str = "" ;
        the while (( STR = br .readline ()) ! = null ) {
            // use the string to match the regular method defined
    Matcher m =            Pattern . compile ( "const-string ([vp] \\ d{1,2}), \" (.*) \" " ).matcher( str );
            if ( m .find()) {
                String tmp = m .group ( 2 );
                IF ( tmp .equals ( "" )) {
                    SB .append ( STR + " \ n- " );
                    Continue ;
                }
                // string escape , filtered off \ (such as \ " do not turn upon acquiring the meaning of \ ", But it shou
    tmp = StringEscapeUtils . UnescapeJava ( tmp ); String Register = m .group ( . 1 ); // Register represents a re




                \" " ;
                String dec = "" ;
                if ( Integer . parseInt ( register .substring( 1 )) > 15 && register .startsWith( "v" )) {
                    // here consider the number of registers if the v register is greater than 15 When using the rang
    dec = " invoke-static/range {" + register + " .. " + register + "}, Lcom/qtfreet00;->decode(Ljava/lang/String
                    Add a decryption method
                 } else if ( register .startsWith( "v" ) || ( register .startsWith( "p" ) && Integer . parseInt ( register .s
                    // here p is at 10 The above (unclear), there will be some problems, because not too much conta
    / p in the method generally represents the input static methods from p0 start from a non-static method p1 s
                    " invoke-static {" + register + "}, Lcom/qtfreet00;->decode(Ljava/lang/String;)Ljava/lang/String
                } else {
                    sb .append( str + " \n " );
                    Continue ;
                }
                String mov = " move-result-object " + register ;
                sb .append( sign + " \n\n " );
                sb .append( dec + " \n\n " );
                sb .Append(Mov + " \n " );
            } else {
                sb .append( str + " \n " );
            }
        }
        br .close();
        read .close();
        // Overwrite the source file
    FileOutputStream fos = new FileOutputStream( new File(path)); fos .write( sb .toString().getBytes( "UTF-8" ))





      }
    }
```

There is no consideration of global variables here. Interested parties can communicate together

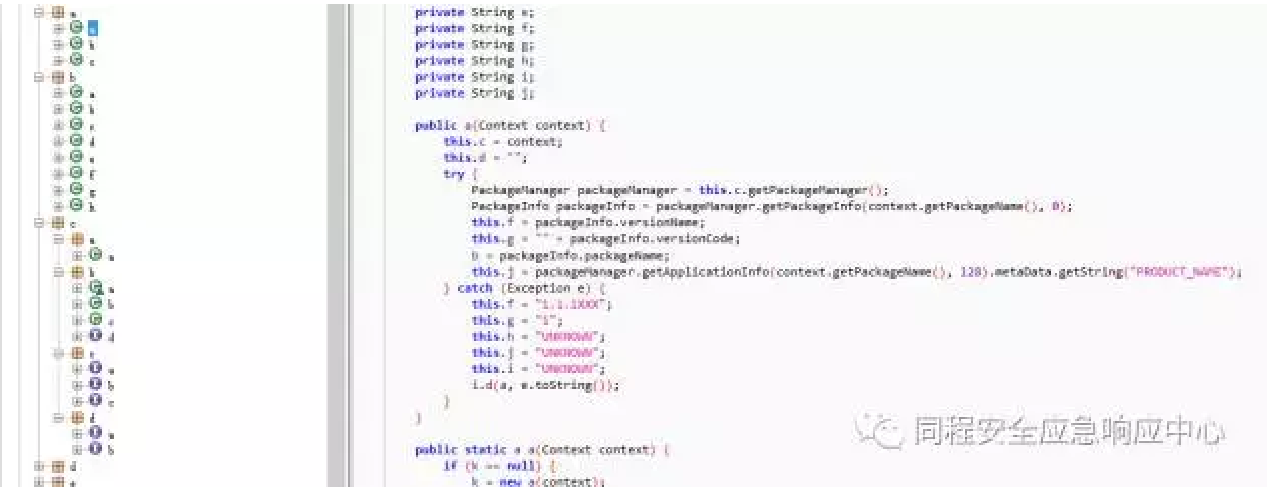
Since you want to use string encryption, the encryption method naturally cannot leak:

```
Public static String encode ( String str) {
   // Get byte array according to default encoding
byte [] bytes = str.getBytes(); int len = bytes . length ; int keyLen = KEY .length(); for ( int i = 0 ; i < len ; i -



      [ I ] ^ KEY .charAt ( I % KEYLEN ));
   }
   the StringBuilder SB = new new the StringBuilder ( bytes . Length * 2 );
   // byte array disassembled into each byte 2 bit 16 binary integers
for ( Int i = 0 ; i < bytes . length ; i ++ ) { sb .append( hexString .charAt(( bytes [ i ]
      & 0xf0 ) >> 4 ));
      sb .append( hexString .charAt(( bytes [ i ] & 0x0f ) >> 0 ));
   }
   return sb .toString();
}
```

Insert the decrypted smali file, here you can write an Android project, write the decryption method, and then package decompiled into smali can, pay attention to the path of the inserted code and the corresponding method, and then use the apktool package signature.

The use of string obfuscation can protect the security of some sensitive information to a greater extent, such as encryption. JAVA generally uses native encryption APIs to specify types in the string, such as "AES/CBC/PKCS5Padding", and some The key is hard coded, and of course it is not difficult to restore this string confusion.

After talking about the string confusion, what is the dictionary confusion? The default confusing effect in ProGuard is as follows:

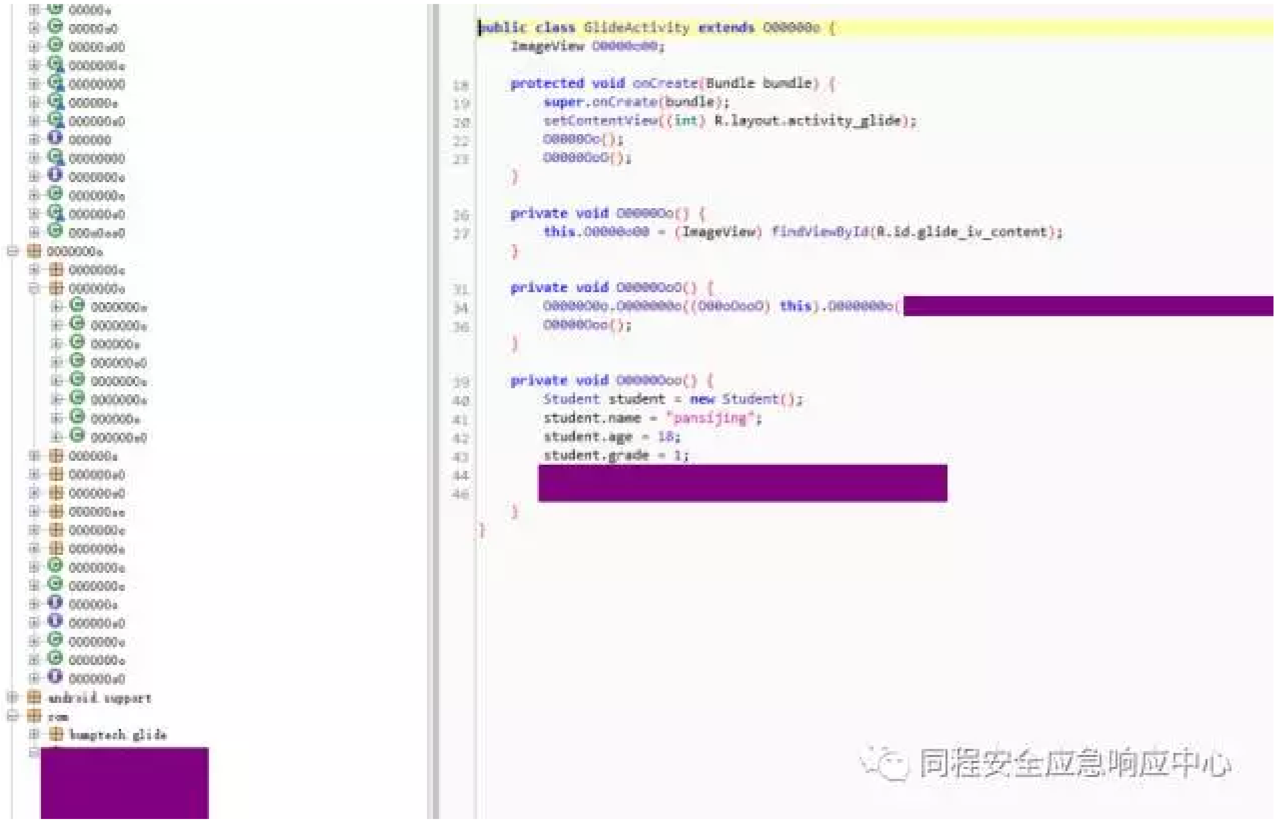


```
private String e;
private String f;
private String g;
private String h;
private String i;
private String j;

public a(Context context) {
    this.c = context;
    this.d = "";
    try {
        PackageManager packageManager = this.c.getPackageManager();
        PackageInfo packageInfo = packageManager.getPackageInfo(context.getPackageName(), 0);
        this.f = packageInfo.versionName;
        this.g = "" + packageInfo.versionCode;
        b = packageInfo.packageName;
        this.j = packageManager.getApplicationInfo(context.getPackageName(), 128).metaData.getString("PRODUCT_NAME");
    } catch (Exception e) {
        this.f = "1.1.1XXX";
        this.g = "1";
        this.h = "UNKNOWN";
        this.j = "UNKNOWN";
        this.i = "UNKNOWN";
        i.d(a, e.toString());
    }
}

public static a a(Context context) {
    if (k == null) {
        k = new a(context);
```

You can see that the variable name, method name, and directory name have been replaced with letters such as abcd, but this confusion can only improve the inverse difficulty to a certain extent, you can semantically or manually recover variable names, how large To increase the difficulty, we need to use a dictionary. In ProGuard, we allow users to use a custom dictionary and provide three commands.

```
-obfuscationdictionary dic.txt

-classobfuscationdictionary dic.txt

-packageobfuscationdictionary dic.txt
```

What about the effect of this custom dictionary? as follows:



```
public class GlideActivity extends O00000o {
    ImageView O0000o00;

    protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView((int) R.layout.activity_glide);
        O00000Oo();
        O00000oO();
    }

    private void O00000Oo() {
        this.O0000o00 = (ImageView) findViewById(R.id.glide_iv_content);
    }

    private void O00000oO() {
        O00000Oo.O00000Oo((O00oOooO) this).O00000Oo([redacted]
        O00000oo();
    }

    private void O00000oo() {
        Student student = new Student();
        student.name = "pansijing";
        student.age = 18;
        student.grade = 1;
        [redacted]
    }
}
```

In normal use, you need to pay attention to the configuration in proguard-rules.pro to prevent the app from running or running abnormally.

Pay attention to the public number and send "obfuscated" to get the code in the text.

Search for "Just safe" or scan the QR code below for YSRC public number.

Past articles
I may have used a fake stealth mode
Hadoop Security Practices for Tongdu Tour
Click on my link to play a face calculator
Click on my link and I will know which chrome plugins you used
YSRC sincerity, patrol - corporate security loophole rapid emergency, cruise system
Android simulator detection based on file characteristics
Android reverse and virus analysis
F-Scrack weak password detection script
CSP Bypass in unsafe mode
Passive scanner GourdScan v2.0 released!
Android App Common Reverse Tools and Tips
XSS Trap - Simple Trial of XSS DNS Protection

A BLACK PATH TOWARD THE SUN - Introduction to the HTTP Tunnel Tool

Windows Fuzzing artifact ---- Winafl

Report