Lab9 Report

Guan Xiao 18307130012

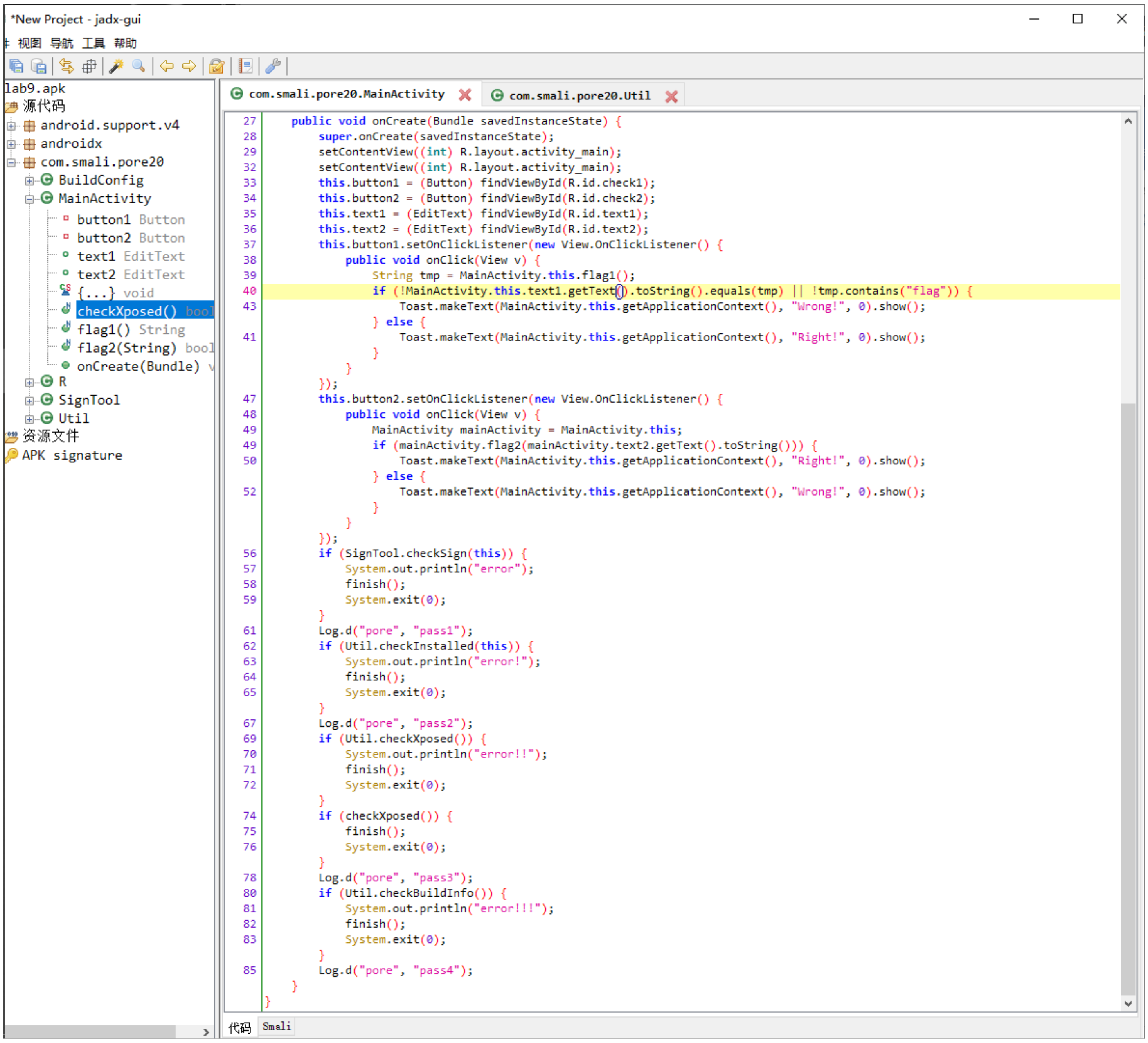
Flag1： flag{f14g_fr03_j1n0o}

Flag2：

How to get the flags & How many anti-debug trick and how to beat them:

① Use apktool to extra the smali

② Use jadx to analyze the java function:

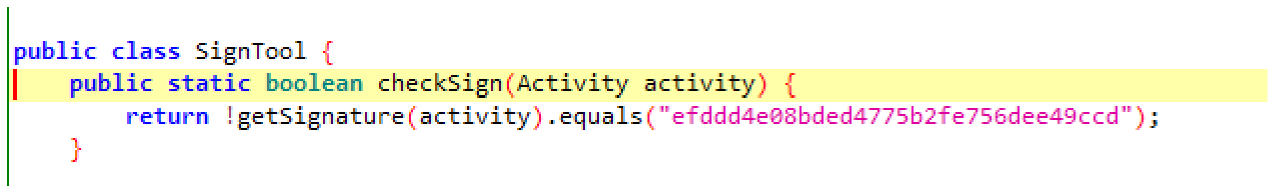


```java
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView((int) R.layout.activity_main);
        setContentView((int) R.layout.activity_main);
        this.button1 = (Button) findViewById(R.id.check1);
        this.button2 = (Button) findViewById(R.id.check2);
        this.text1 = (EditText) findViewById(R.id.text1);
        this.text2 = (EditText) findViewById(R.id.text2);
        this.button1.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                String tmp = MainActivity.this.flag1();
                if (!MainActivity.this.text1.getText().toString().equals(tmp) || !tmp.contains("flag")) {
                    Toast.makeText(MainActivity.this.getApplicationContext(), "Wrong!", 0).show();
                } else {
                    Toast.makeText(MainActivity.this.getApplicationContext(), "Right!", 0).show();
                }
            }
        });
        this.button2.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                MainActivity mainActivity = MainActivity.this;
                if (mainActivity.flag2(mainActivity.text2.getText().toString())) {
                    Toast.makeText(MainActivity.this.getApplicationContext(), "Right!", 0).show();
                } else {
                    Toast.makeText(MainActivity.this.getApplicationContext(), "Wrong!", 0).show();
                }
            }
        });
        if (SignTool.checkSign(this)) {
            System.out.println("error");
            finish();
            System.exit(0);
        }
        Log.d("pore", "pass1");
        if (Util.checkInstalled(this)) {
            System.out.println("error!");
            finish();
            System.exit(0);
        }
        Log.d("pore", "pass2");
        if (Util.checkXposed()) {
            System.out.println("error!!");
            finish();
            System.exit(0);
        }
        if (checkXposed()) {
            finish();
            System.exit(0);
        }
        Log.d("pore", "pass3");
        if (Util.checkBuildInfo()) {
            System.out.println("error!!!");
            finish();
            System.exit(0);
        }
        Log.d("pore", "pass4");
    }
}
```

We can find there're two button and four anti-debug check.

③ Analyze the four anti-debug check function.

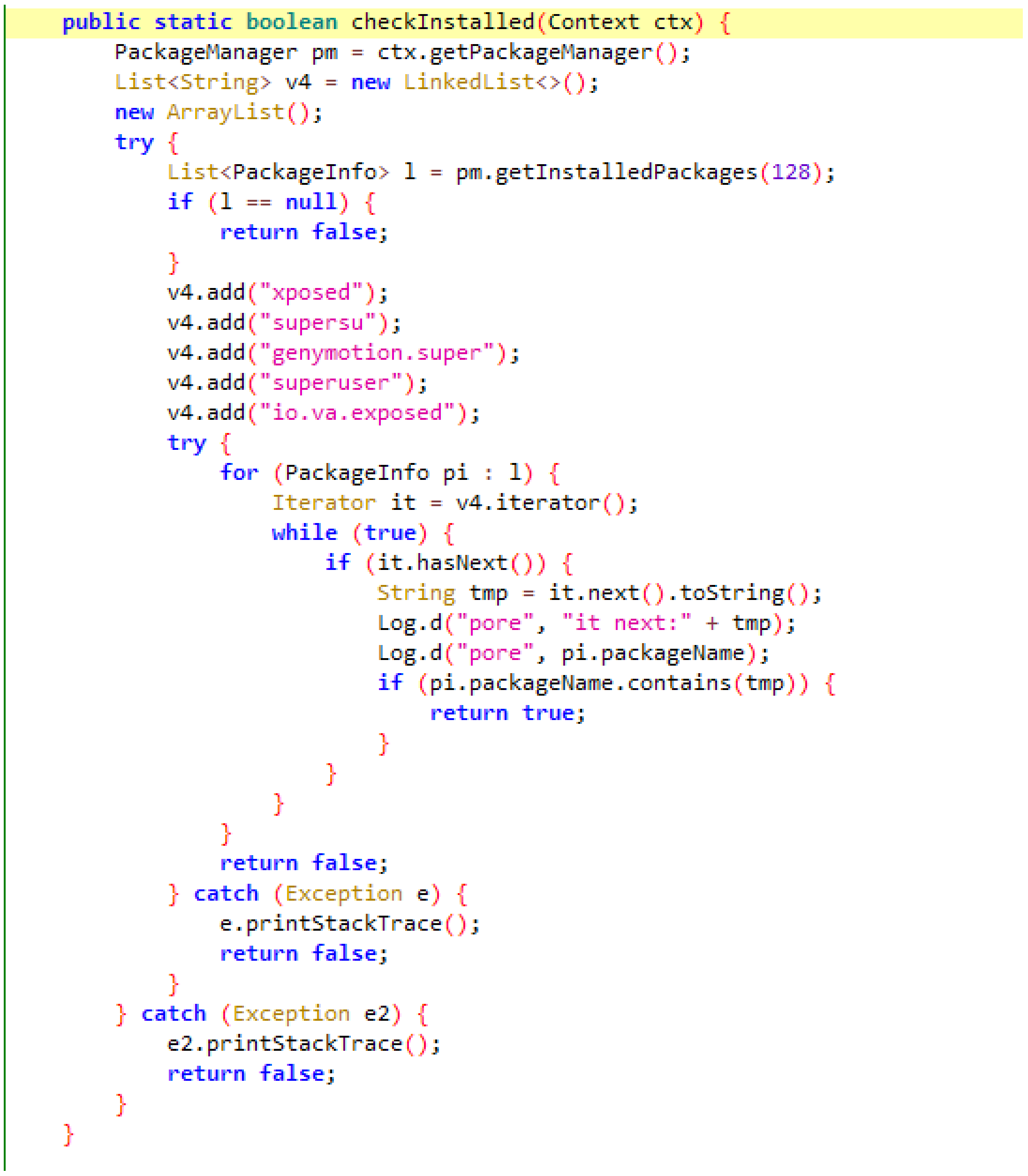
First is check whether the apk had been re-packaged.



```java
public class SignTool {
    public static boolean checkSign(Activity activity) {
        return !getSignature(activity).equals("efddd4e08bded4775b2fe756dee49ccd");
    }
```

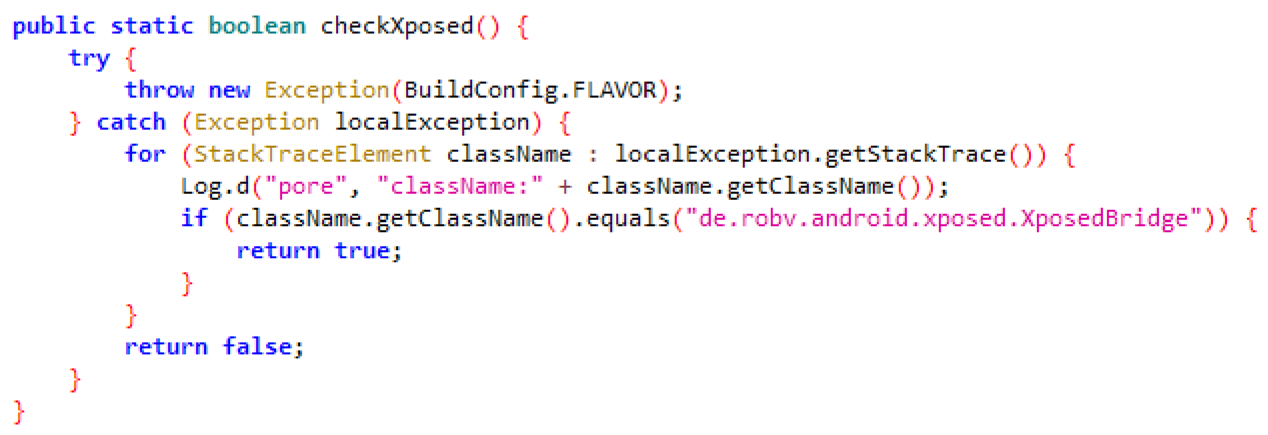
We never change the apk, so ignore.

Second is check whether there's Xposed or superuser by packageManager.



```java
    public static boolean checkInstalled(Context ctx) {
        PackageManager pm = ctx.getPackageManager();
        List<String> v4 = new LinkedList<>();
        new ArrayList();
        try {
            List<PackageInfo> l = pm.getInstalledPackages(128);
            if (l == null) {
                return false;
            }
            v4.add("xposed");
            v4.add("supersu");
            v4.add("genymotion.super");
            v4.add("superuser");
            v4.add("io.va.exposed");
            try {
                for (PackageInfo pi : l) {
                    Iterator it = v4.iterator();
                    while (true) {
                        if (it.hasNext()) {
                            String tmp = it.next().toString();
                            Log.d("pore", "it next:" + tmp);
                            Log.d("pore", pi.packageName);
                            if (pi.packageName.contains(tmp)) {
                                return true;
                            }
                        }
                    }
                }
                return false;
            } catch (Exception e) {
                e.printStackTrace();
                return false;
            }
        } catch (Exception e2) {
            e2.printStackTrace();
            return false;
        }
    }
```

Third is to check Xposed by throwing an error and check whether there's Xposed in the error stack.



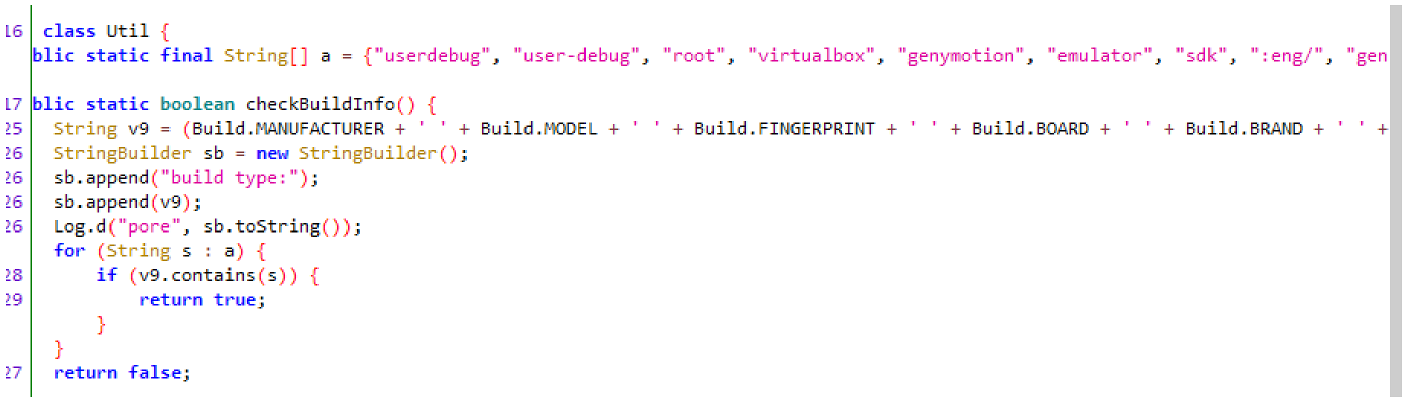
```
public static boolean checkXposed() {
    try {
        throw new Exception(BuildConfig.FLAVOR);
    } catch (Exception localException) {
        for (StackTraceElement className : localException.getStackTrace()) {
            Log.d("pore", "className:" + className.getClassName());
            if (className.getClassName().equals("de.robv.android.xposed.XposedBridge")) {
                return true;
            }
        }
        return false;
    }
}
```

We can bypass this check by prevent them to find Xposed.

We use a Xposed Plugin called “Xposed Hider” here.

Fourth is to check the OS information to check whether it’s running in a simulator by trying to find

some string in the OS information.



```
 class Util {
blic static final String[] a = {"userdebug", "user-debug", "root", "virtualbox", "genymotion", "emulator", "sdk", ":eng/", "gen

blic static boolean checkBuildInfo() {
  String v9 = (Build.MANUFACTURER + ' ' + Build.MODEL + ' ' + Build.FINGERPRINT + ' ' + Build.BOARD + ' ' + Build.BRAND + ' ' +
  StringBuilder sb = new StringBuilder();
  sb.append("build type:");
  sb.append(v9);
  Log.d("pore", sb.toString());
  for (String s : a) {
      if (v9.contains(s)) {
          return true;
      }
  }
  return false;
```

We can bypass the check by changing the OS information.

We use a Xposed Plugin called “android ID changer” here.

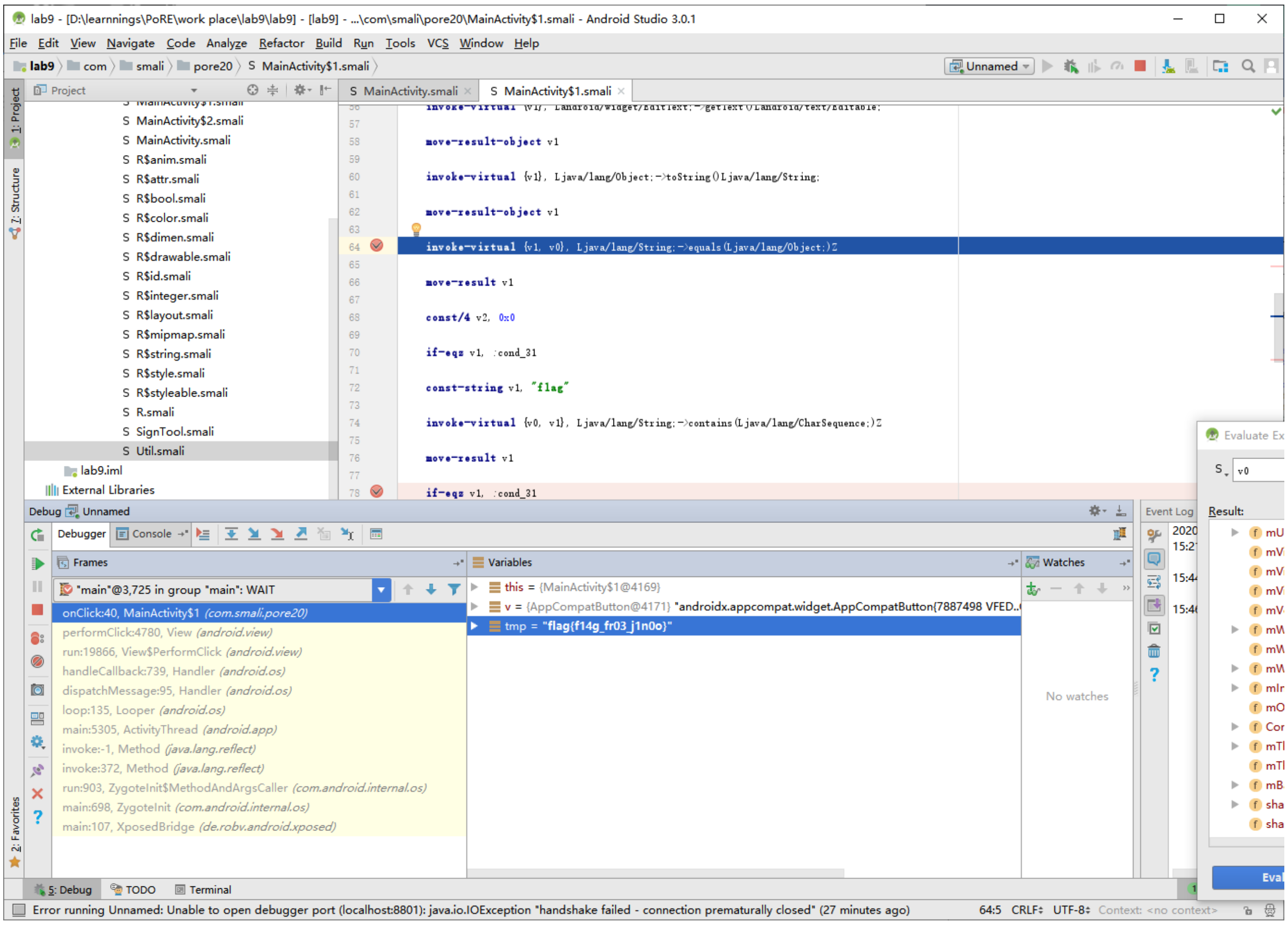
Then we can normally open the apk, and debug it.

For Task1:

We breakpoint the function before it compare our input and flag1.

We can easily find flag1 in the variables stack.

It’s “flag{f14g_fr03_j1n0o}”.


lab9 - [D:\learnnings\PoRE\work place\lab9\lab9] - [lab9] - ...\com\smali\pore20\MainActivity$1.smali - Android Studio 3.0.1

```
invoke-virtual {v1}, Landroid/widget/EditText;->getText()Landroid/text/Editable;

move-result-object v1

invoke-virtual {v1}, Ljava/lang/Object;->toString()Ljava/lang/String;

move-result-object v1

invoke-virtual {v1, v0}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v1

const/4 v2, 0x0

if-eqz v1, :cond_31

const-string v1, "flag"

invoke-virtual {v0, v1}, Ljava/lang/String;->contains(Ljava/lang/CharSequence;)Z

move-result v1

if-eqz v1, :cond_31
```

Frames:

```
"main"@3,725 in group "main": WAIT
onClick:40, MainActivity$1 (com.smali.pore20)
performClick:4780, View (android.view)
run:19866, View$PerformClick (android.view)
handleCallback:739, Handler (android.os)
dispatchMessage:95, Handler (android.os)
loop:135, Looper (android.os)
main:5305, ActivityThread (android.app)
invoke:-1, Method (java.lang.reflect)
invoke:372, Method (java.lang.reflect)
run:903, ZygoteInit$MethodAndArgsCaller (com.android.internal.os)
main:698, ZygoteInit (com.android.internal.os)
main:107, XposedBridge (de.robv.android.xposed)
```

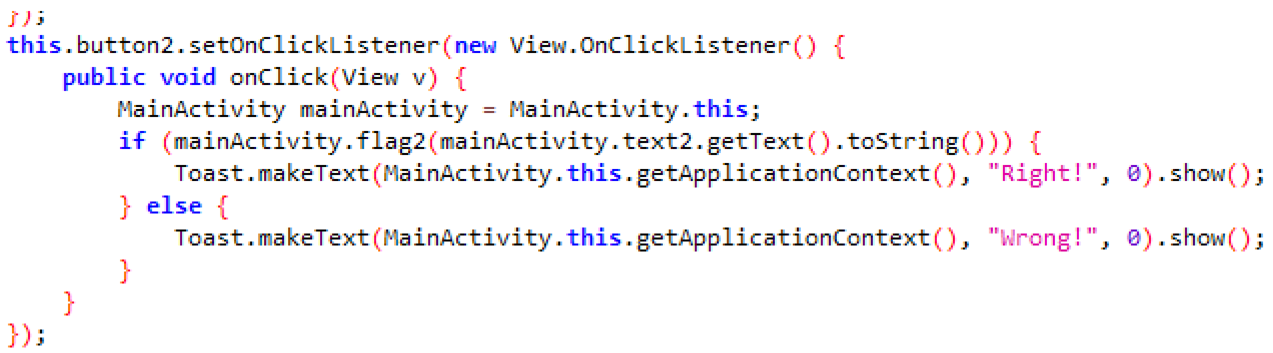
Variables:

```
this = {MainActivity$1@4169}
v = {AppCompatButton@4171} "androidx.appcompat.widget.AppCompatButton{7887498 VFED..
tmp = "flag{f14g_fr03_j1n0o}"
```

```
Error running Unnamed: Unable to open debugger port (localhost:8801): java.io.IOException "handshake failed - connection prematurally closed" (27 minutes ago)
```

For Task2:

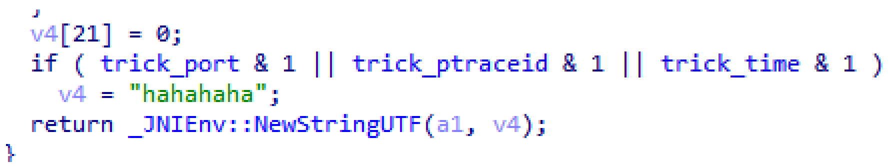
① Find logic in java is to call the native method to compare the string. So we can only use IDA to crack the flag2.



```
});
this.button2.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        MainActivity mainActivity = MainActivity.this;
        if (mainActivity.flag2(mainActivity.text2.getText().toString())) {
            Toast.makeText(MainActivity.this.getApplicationContext(), "Right!", 0).show();
        } else {
            Toast.makeText(MainActivity.this.getApplicationContext(), "Wrong!", 0).show();
        }
    }
});
```

② We must crack the libnative-lib.so, using IDA to analyze it.

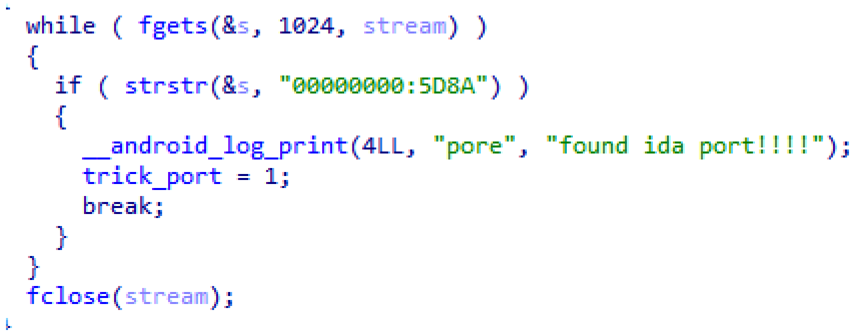
We can find in the .so that was many encrypted function.



```
  v4[21] = 0;
  if ( trick_port & 1 || trick_ptraceid & 1 || trick_time & 1 )
    v4 = "hahahaha";
  return _JNIEnv::NewStringUTF(a1, v4);
}
```

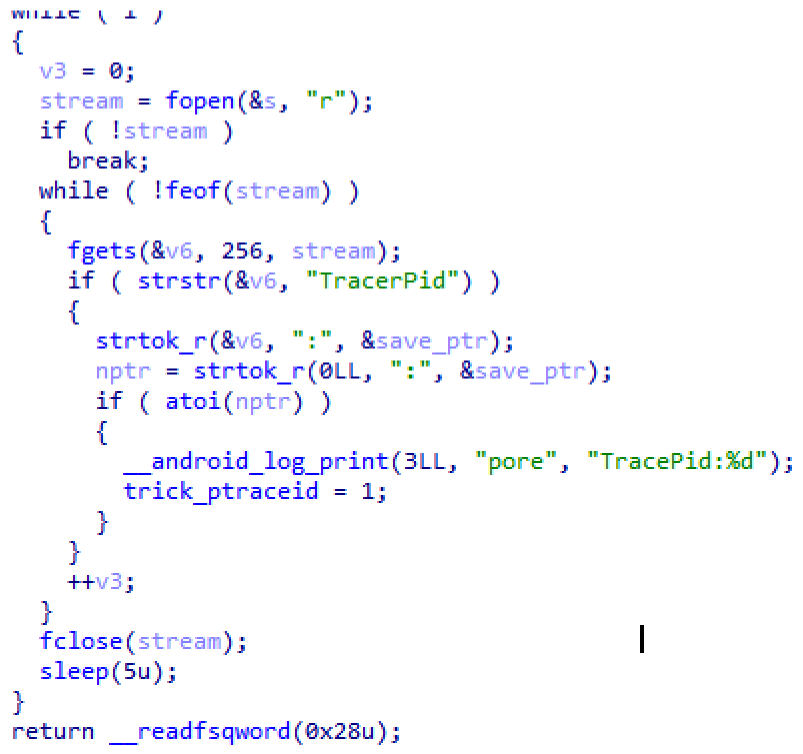
We can find this in function "Java_com_smali_pore20_MainActivity_flag1", which show us there will be three kinds of check to anti-IDA reversion.

Trick_port is a function looks for 23946 port in android's tcp port list. Because the default port of IDA native reversion is 23946. When this port can be found, means that the apk is being debugging.



```
  while ( fgets(&s, 1024, stream) )
  {
    if ( strstr(&s, "00000000:5D8A") )
    {
      __android_log_print(4LL, "pore", "found ida port!!!!");
      trick_port = 1;
      break;
    }
  }
  fclose(stream);
```

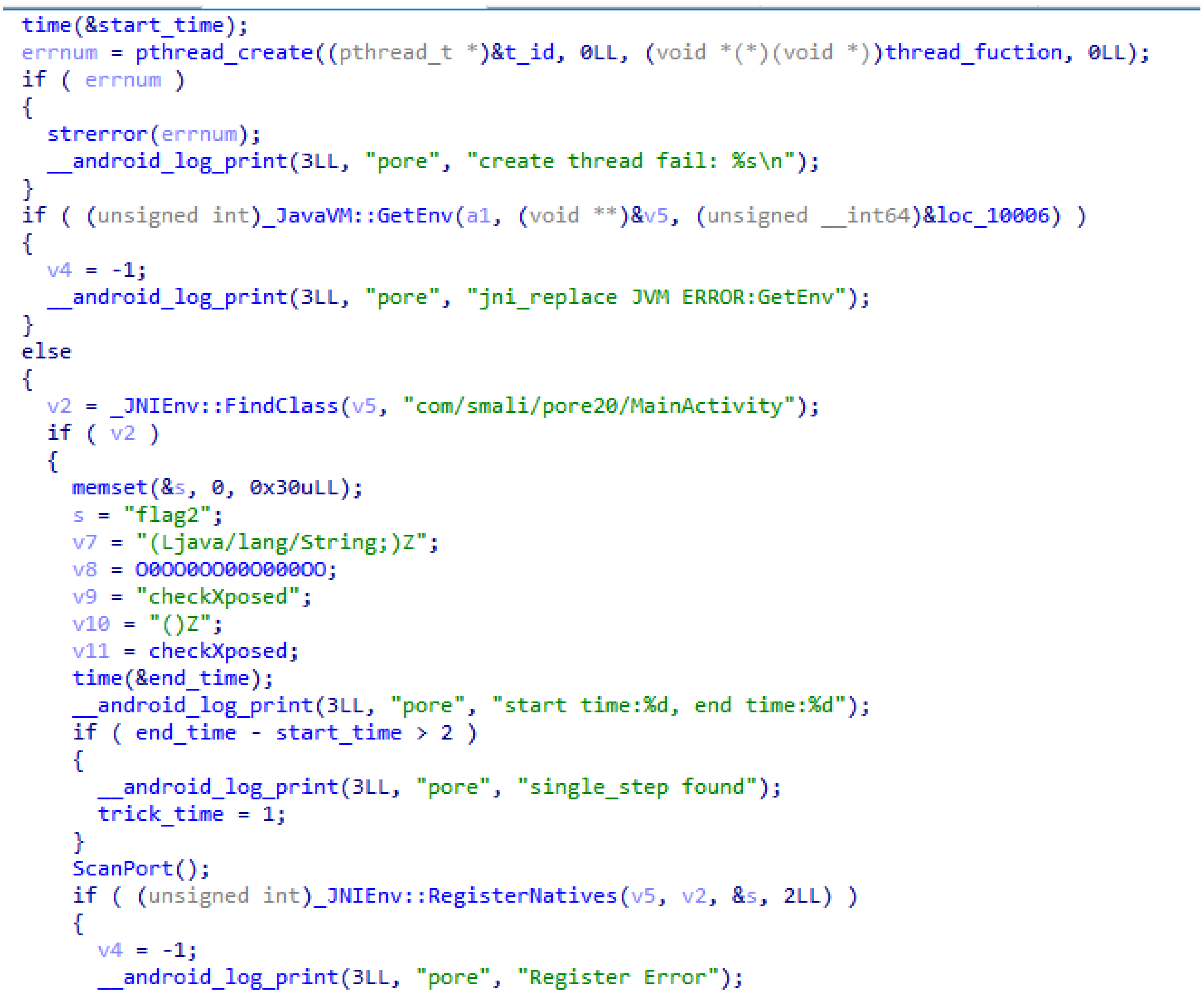
Trick_ptraceid is a function that check the pid, if pid isn't zero, confirm it's being traced.



```
while ( 1 )
{
  v3 = 0;
  stream = fopen(&s, "r");
  if ( !stream )
    break;
  while ( !feof(stream) )
  {
    fgets(&v6, 256, stream);
    if ( strstr(&v6, "TracerPid") )
    {
      strtok_r(&v6, ":", &save_ptr);
      nptr = strtok_r(0LL, ":", &save_ptr);
      if ( atoi(nptr) )
      {
        __android_log_print(3LL, "pore", "TracePid:%d");
        trick_ptraceid = 1;
      }
    }
    ++v3;
  }
  fclose(stream);
  sleep(5u);
}
return __readfsqword(0x28u);
```

Trick_time is a function to check the running time of JNI_OnLoad to prevent single-step debug.

Using a time stamp to record start time, and calculate the time between start and end.

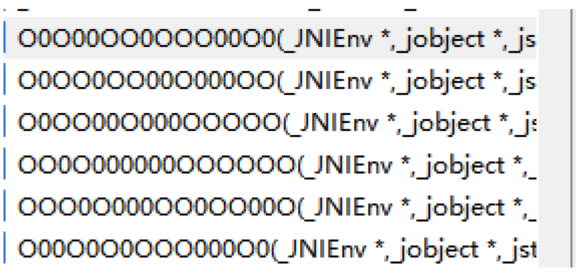


```
time(&start_time);
errnum = pthread_create((pthread_t *)&t_id, 0LL, (void *(*)(void *))thread_fuction, 0LL);
if ( errnum )
{
  strerror(errnum);
  __android_log_print(3LL, "pore", "create thread fail: %s\n");
}
if ( (unsigned int)_JavaVM::GetEnv(a1, (void **)&v5, (unsigned __int64)&loc_10006) )
{
  v4 = -1;
  __android_log_print(3LL, "pore", "jni_replace JVM ERROR:GetEnv");
}
else
{
  v2 = _JNIEnv::FindClass(v5, "com/smali/pore20/MainActivity");
  if ( v2 )
  {
    memset(&s, 0, 0x30uLL);
    s = "flag2";
    v7 = "(Ljava/lang/String;)Z";
    v8 = O00000000000O0O;
    v9 = "checkXposed";
    v10 = "()Z";
    v11 = checkXposed;
    time(&end_time);
    __android_log_print(3LL, "pore", "start time:%d, end time:%d");
    if ( end_time - start_time > 2 )
    {
      __android_log_print(3LL, "pore", "single_step found");
      trick_time = 1;
    }
    ScanPort();
    if ( (unsigned int)_JNIEnv::RegisterNatives(v5, v2, &s, 2LL) )
    {
      v4 = -1;
      __android_log_print(3LL, "pore", "Register Error");
```

One way to block this function is to change it's machine code in order to change it's running method.

We can use 010Editor to modified this .so, such as changed this 1 to 0 to ensure the check result is always false.

③ And We can find the .so use rename confusion to confuse the reverser.



```
O0O000O0O0O00O0(_JNIEnv *,_jobject *,_js
O0O0O0O00O000OO(_JNIEnv *,_jobject *,_js
O0O00O0O00OOOOO(_JNIEnv *,_jobject *,_js
O0O0O00000OOOOOO(_JNIEnv *,_jobject *,_
O0O0O000O0O0O00O(_JNIEnv *,_jobject *,_
O00O0O0O0OO00O0O(_JNIEnv *,_jobject *,_jst
```

④ Also the nested function call is used.