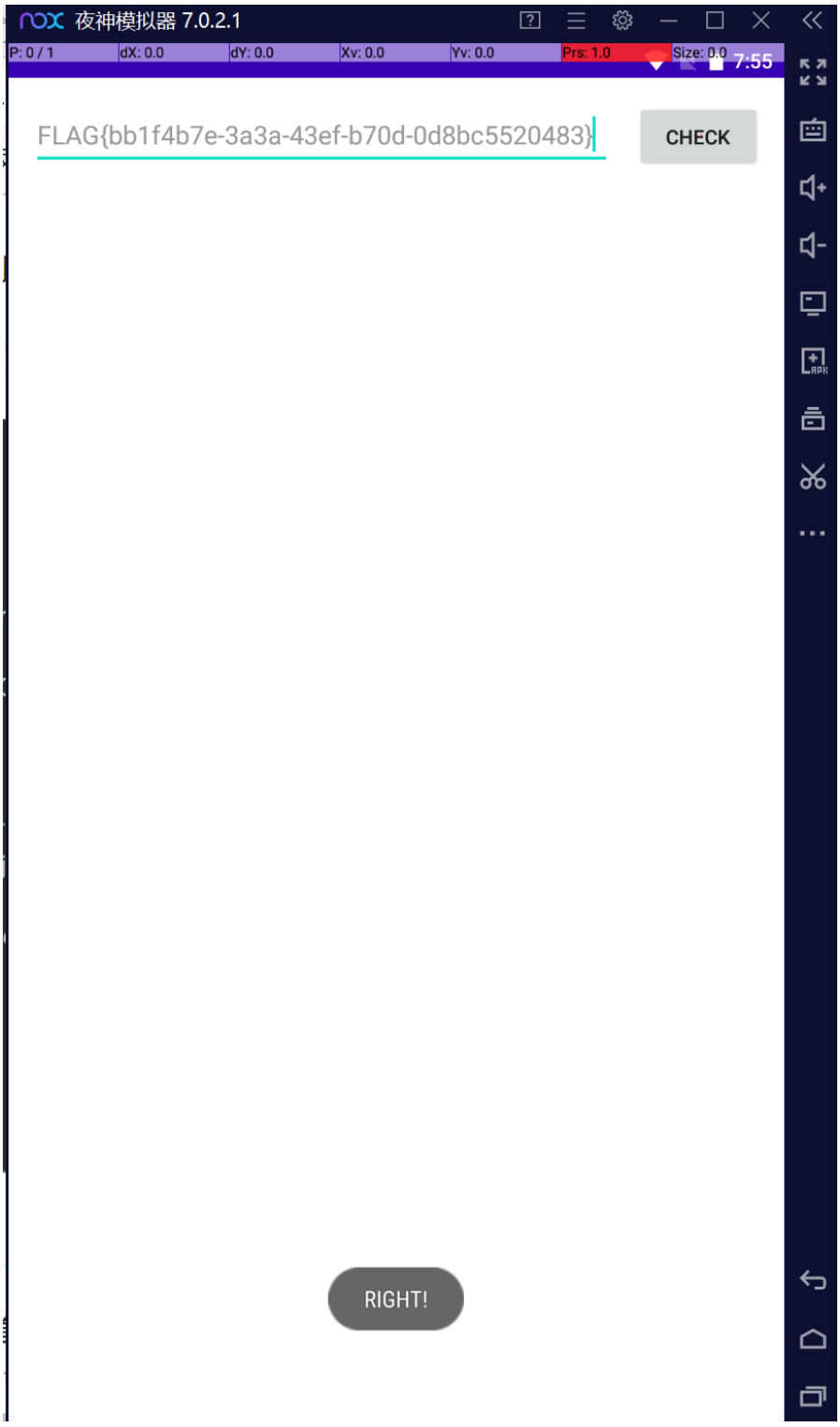
Lab9 Unpacking

STU ID: 20307130350
Your Flag: FLAG{bb1f4b7e-3a3a-43ef-b70d-0d8bc5520483}



Analysis Process Breakdown:

1. How this app is hardened and packed? Please explain the logic in as much detail as possible.

Dex 加壳需要三个对象：

1. 源程序：需要加壳处理的被保护代码
2. 解壳程序：解密解壳数据，运行时通过 DexClassLoader 动态加载
3. 加壳程序：加密源程序为解壳数据、组装解壳程序和解壳数据。

加壳过程：

1. 加密源程序 apk 文件为解壳数据（dex 文件）
2. 把解壳数据放在某个位置
3. 修改解壳程序 DEX 头中的 checksum，signature 和 file_size 等头信息
4. 修改源程序 manifest 文件并覆盖解壳程序 manifest 文件

5. 解密过程：

宿主 apk 启动——>宿主 Application 解密 Apk——>替换 ClassLoader——>替换资源路径——>替换 Application 对象。

A）解壳代码需要第一时间执行：

解壳程序必须在 Android 系统启动组件之前运行，完成对解壳数据的解壳及 apk 文件的动态加载，否则会使程序出现加载类失败的异常。因为 application 会首先调用，因此，在 manifest 文件中对 application 的配置中实现解壳代码的第一时间运行。

```
" android:versionCode="1" android:versionName="1.0" android:compileSdkVersion="23" android:compileSdkVersionCode
30"/>
l="@string/app_name" android:icon="@mipmap/ic_launcher" android:name="com.android.dexshell.ProxyApplication" and
lue="com.android.sourceapp.XApplication"/>
 android:exported="true">
```

B）替换回源程序原有的 application：

在解壳代码运行完成之后，替换回源程序原有的 application 对象。通过在 manifest 文件中配置原有的 application 类信息，使得解壳程序运行完毕时能通过创建配置的 application 对象，并通过反射修改回原 application。

```
uses-sdk android:minSdkVersion="19" android:targetSdkVersion="30"/>
application android:theme="@style/Theme.Dexshell" android:label="@string/app
    <meta-data android:name="APPLICATION_CLASS_NAME" android:value="com.andro
    <activity android:name="com.android.sourceapp.MainActivity" android:expor
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
/application>
```

```
85  public void onCreate() {
86      super.onCreate();
89      String srcAppClassName = "";
        try {
93          Bundle bundle = getPackageManager().getApplicationInfo(getPackageName(), 128).metaData;
94          if (bundle == null || !bundle.containsKey("APPLICATION_CLASS_NAME")) {
98              Log.d("demo", "can not find the information of application");
99              return;
            }
95          srcAppClassName = bundle.getString("APPLICATION_CLASS_NAME");
106         Object currentActivityThread = RefInvoke.invokeStaticMethod("android.app.ActivityThread", "currentActivityThread", new Class[0], new Object[0]);
108         Object mBoundApplication = RefInvoke.getFieldOjbect("android.app.ActivityThread", currentActivityThread, "mBoundApplication");
110         Object loadedApkInfo = RefInvoke.getFieldOjbect("android.app.ActivityThread$AppBindData", mBoundApplication, "info");
113         RefInvoke.setFieldOjbect("android.app.LoadedApk", "mApplication", loadedApkInfo, (Object) null);
121         ((ArrayList) RefInvoke.getFieldOjbect("android.app.ActivityThread", currentActivityThread, "mAllApplications")).remove(RefInvoke.getFieldOjbect("
125         ((ApplicationInfo) RefInvoke.getFieldOjbect("android.app.LoadedApk", loadedApkInfo, "mApplicationInfo")).className = srcAppClassName;
126         ((ApplicationInfo) RefInvoke.getFieldOjbect("android.app.ActivityThread$AppBindData", mBoundApplication, "appInfo")).className = srcAppClassName;
128         Application app = (Application) RefInvoke.invokeMethod("android.app.LoadedApk", "makeApplication", loadedApkInfo, new Class[]{Boolean.TYPE, Instr
133         RefInvoke.setFieldOjbect("android.app.ActivityThread", "mInitialApplication", currentActivityThread, app);
            for (Object providerClientRecord : ((ArrayMap) RefInvoke.getFieldOjbect("android.app.ActivityThread", currentActivityThread, "mProviderMap")).val
141             RefInvoke.setFieldOjbect("android.content.ContentProvider", "mContext", RefInvoke.getFieldOjbect("android.app.ActivityThread$ProviderClientRe
            }
144         app.onCreate();
        } catch (Exception e) {
103         Log.d("demo", "can not find the package manager");
        }
```

C) 使解壳后的 apk 资源文件被代码动态引用：

代码默认引用的资源文件在最外层的解渴程序中，因此要增加系统的资源加载路径来实现对解壳后 apk 文件资源的加载。

```
33  public class ProxyApplication extends Application {
        final String TAG = "APPLICATION_CLASS_NAME";
        private String mDexAbsolutePath;
        private String mLibAbsolutePath;
        private String mSrcApkAbsolutePath;

        private native byte[] getDexFileFromShellApk();

        private native void releaseSrcApkAndSrcLibFiles(byte[] bArr);

        static {
35          System.loadLibrary("dexshell");
        }

        /* access modifiers changed from: protected */
40      public void attachBaseContext(Context base) {
41          super.attachBaseContext(base);
44          File odex = getDir("payload_odex", 0);
45          File libs = getDir("payload_lib", 0);
46          this.mDexAbsolutePath = odex.getAbsolutePath();
47          this.mLibAbsolutePath = libs.getAbsolutePath();
48          this.mSrcApkAbsolutePath = odex.getAbsolutePath() + "/payload.apk";
49          File srcApkFile = new File(this.mSrcApkAbsolutePath);
50          if (!srcApkFile.exists()) {
                try {
53                  srcApkFile.createNewFile();
                } catch (IOException e) {
55                  e.printStackTrace();
                }
66              releaseSrcApkAndSrcLibFiles(getDexFileFromShellApk());
```

1. 创建 payload_odex 和 payload_lib 文件，分别用来放置源 apk（源 dex 文件）和.so 文件。
2. This.mDexAbosolutePath 用来存放源 apk 释放出来的 dex；
3. mLibAbsolutePath 用来存放源 apk 用到的 so 文件
4. mSrcApkAbusolutePath 用来存放解密后的 apk。

D）通过 DexClassLoader 实现对 apk 代码的动态加载：

DexClassLoader 加载的类是没有组件生命周期，因此，即使 DexClassLoader 通过对 apk 的动态加载完成了对组件类的加载，当系统启动该组件时，还会出现加载类失败的异常。这是因源代码的组件类的加载是由另一个 ClassLoader 加载的，DexClassLoader 与系统组件的 ClassLoader 不存在关系，系统组件 ClassLoader 自然找不到由 DexClassLoader 加载的类。但是，如果将系统组件 ClassLoader 的 parent 修改为 DexClassLoader，就可以实现对 apk 代码的动态加载——双亲委托机制。

```
Object currentActivityThread = RefInvoke.invokeStaticMethod("android.app.ActivityThread", "currentActivityThread", new Class[0], new Object[0]);
WeakReference weakReference = (WeakReference) ((ArrayMap) RefInvoke.getFieldOjbect("android.app.ActivityThread", currentActivityThread, "mPackages")).get
DexClassLoader newDexClassLoader = new DexClassLoader(this.mSrcApkAbsolutePath, this.mDexAbsolutePath, this.mLibAbsolutePath, (ClassLoader) RefInvoke.getI
RecursiveDeleteFile(odex);
RecursiveDeleteFile(libs);
RefInvoke.setFieldOjbect("android.app.LoadedApk", "mClassLoader", weakReference.get(), newDexClassLoader);
```

配置动态加载环境：

1. 反射获取主线程对象，并从中获取所有已加载的 package 信息，并找到当前 loadapk 的弱引用。
   a) 获取主线程对象。CurrentActivityThread
      i. `Object currentActivityThread = RefInvoke.invokeStaticMethod("android.app.ActivityThread", "currentActivityThread", new Class[0], new Object[0]);`
   b) 获取 loadapk 的弱引用：weakReference
      i. `WeakReference weakReference = (WeakReference) ((ArrayMap) RefInvoke.getFieldOjbect("android.app.ActivityThread", currentActivityThread, "mPackages"))`
2. 用于加载源 Apk，传入 apk 路径、dex 释放路径，so 路径，其父节点的 DexClassLoader——双亲委托机制
   a) `(ClassLoader) RefInvoke.getFieldOjbect("android.app.LoadedApk", weakReference.get(), "mClassLoader"));`
3. 创建一个新的 DexClassLoader 对象，指定 apk 路径、odex 路径和 lib 路径。
   a) `DexClassLoader newDexClassLoader = new DexClassLoader(this.mSrcApkAbsolutePath, this.mDexAbsolutePath, this.mLibAbsolutePath, (ClassLoader) RefInvo`
4. 加载完源 apk 之后删除文件
   a)
   ```
   RecursiveDeleteFile(odex);
   RecursiveDeleteFile(libs);
   ```
5. 将父节点 DexClassLoader 替换：
   a) `RefInvoke.setFieldOjbect("android.app.LoadedApk", "mClassLoader", weakReference.get(), newDexClassLoader);`

2. How do you manage to get the source app?

A) 根据 1 的分析，先用 Xposed hook attachBaseContext 方法。在日志中确实可以看到成功 hook 到代理类的 attach 方法。

a)
```
}
XposedBridge.log( text: "ProxyAppication has hooked! ");
XposedHelpers.findAndHookMethod(clazz, methodName: "attachBaseContext", Context.class, (XC_MethodHook) afte
        Context context = (Context) param.args[0];
        ClassLoader classLoader = context.getClassLoader();

        XposedBridge.log( text: "hook到attach方法");
        XposedBridge.log( text: "得到"+classLoader);
        hookApkFile(loadPackageParam);
```

B) 根据 1 的分析，在 attachBaseContext 中加载了源 dex 文件，但是之后就被删除。因此 hook 该方法并不能 dump 出源 dex 文件。于是改为 hook RecursiveDeleteFile 方法。

```
}

private void RecursiveDeleteFile(File file) {
    if (file.isFile()) {
        file.delete();
    } else if (file.isDirectory()) {
        File[] childFile = file.listFiles();
        boolean z = true;
        boolean z2 = childFile == null;
        if (childFile.length != 0) {
            z = false;
        }
        if (z || z2) {
            file.delete();
            return;
        }
        for (File f : childFile) {
            RecursiveDeleteFile(f);
        }
        file.delete();
    }
}
```

C) 根据 RecursiveDeleteFile 方法的逻辑：如果是文件直接删除，是目录则遍历文件数组一一删除每个文件。

D) 确定 hook 的逻辑：拿到 RecursiveDeleteFile 方法的参数之后（待删除的文件）：
如果文件存在，遍历文件数组：如果是文件，直接创建输入输出流复制；如果是目录，先创建同名目录，再对该目录重复上述操作（遍历，复制）——递归。
用 copyFile 处理该过程：

```
public static void copyFile(File srcDir,File destDir){
    File[] files = srcDir.listFiles();
    if(files==null){
        XposedBridge.log( text: "null");
        return;
    }
    for (File file :files) {
        if(file.isFile()){
            File destFile = new File( pathname: destDir + "/" + file.getName());
            XposedBridge.log( text: file.getName()+" copy!");
            BufferedInputStream bufferedInputStream=null;
            BufferedOutputStream bufferedOutputStream=null;
            try {
                bufferedInputStream = new BufferedInputStream(new FileInputStream(file));
                bufferedOutputStream=new BufferedOutputStream(new FileOutputStream(destFile));
                byte[] buffer=new byte[4096];
                int readLen;
                while((readLen=bufferedInputStream.read(buffer))!=-1){
                    bufferedOutputStream.write(buffer, off: 0,readLen);
                }
                XposedBridge.log( text: file.getName()+" has been copied!");
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }finally {
```

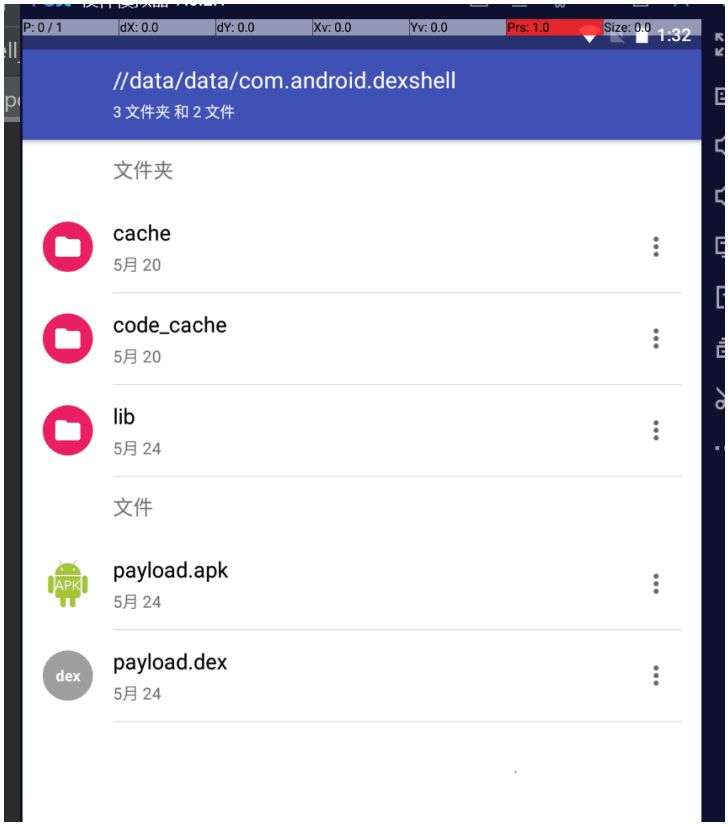
```
                    bufferedOutputStream.write(buffer, off: 0,readLen);
                }
                XposedBridge.log( text: file.getName()+" has been copied!");
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }finally {
                try {
                    bufferedOutputStream.close();
                    bufferedInputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        else{
            File copyDestDir = new File( pathname: destDir + "/" + file.getName());
            copyDestDir.mkdir();
            copyFile(file,copyDestDir);
        }
    }

}
```

Hook 方法：

```
protected void beforeHookedMethod(MethodHookParam param) throws Throwable {
    super.beforeHookedMethod(param);
    Log.e(TAG, msg: "hook RecursiveDeleteFile");
    File file = (File) param.args[0];
    XposedBridge.log( text: file.getPath() + "\t" + file.getName());
    if(file.isDirectory()){
        XposedBridge.log( text: file.getName()+"exists!");
        File[] files = file.listFiles();
        if(files!=null){
            for (File file1 :files) {
                XposedBridge.log( text: file1.getPath()+"\t"+file1.getName());
            }
        }
        copyFile(file,dest);
        XposedBridge.log( text: file.getName()+" has been copied!");
        File[] destFiles = dest.listFiles();
        XposedBridge.log( text: dest.getPath()+"\t"+dest.getName());
        if(destFiles!=null){
            for (File file1 :destFiles) {
                XposedBridge.log( text: file1.getPath()+"\t"+file1.getName());
            }
        }else {
            XposedBridge.log( text: "NULL");
        }
    }else {
        XposedBridge.log( text: file.getName()+" is not a directory!");
    }
```

E) 成功 hook 之后将 nox 中的源 apk 添加到 pc 中。



3. How do you analyze the source app and get the flag?

A） 用 jadx 打开源 apk，搜索“wrong”,可直接定位到加密函数。

```
public void onClick(View view) {
    Context context;
    String str;
    byte[] bytes = this.f1879b.getText().toString().getBytes();
    StringBuilder sb = new StringBuilder();
    int length = bytes.length % 3;
    String str2 = "";
    if (length > 0) {
        bytes = Arrays.copyOf(bytes, (bytes.length + 3) - length);
        while (length < 3) {
            str2 = C0391d.m1402c(str2, "-");
            length++;
        }
    }
    for (int i = 0; i < bytes.length; i += 3) {
        byte b = bytes[i];
        byte b2 = bytes[i + 1];
        byte b3 = bytes[i + 2];
        sb.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt((b >> 2) & 63));
        sb.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt(((b << 4) | (b2 >> 4))
        sb.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt(((b2 << 2) | (b3 >> 6))
        sb.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt(b3 & 63));
    }
    String sb2 = sb.toString();
    if ((sb2.substring(0, sb2.length() - str2.length()) + str2).equals("Zt/aZPBpn5J2vymPf3Yun7v6d7ruflnBn5DOfhYOfN6
        context = MainActivity.this.getApplicationContext();
        str = "RIGHT!";
    } else {
        context = MainActivity.this.getApplicationContext();
        str = "WRONG!";
    }
    Toast.makeText(context, str, 0).show();
}


ublic void onCreate(Bundle bundle) {
```

B)根据 Right 的判断条件和 str2 只能是“”或者带“-“的特征，可以确定 length=0，即 bytes.length 是 3 的倍数。

```
ing sb2 = sb.toString();
((sb2.substring(0, sb2.length() - str2.length()) + str2).equals("Zt/aZPBpn5J2vymPf3Yun7v6d7ruflnBn5DOfhYOfN6pnuWMX5TYwl
context = MainActivity.this.getApplicationContext();
```

```
StringBuilder sb = new StringBuilder();
int length = bytes.length % 3;
String str2 = "";
if (length > 0) {
    bytes = Arrays.copyOf(bytes, (bytes.length + 3) - length);
    while (length < 3) {
        str2 = C0391d.m1402c(str2, "-");
        length++;
    }
}
```

C) 分析 sb2 的形成过程，考虑到 byte 占一字节。可以知道 bytes 的每隔三位 b[3]确定 sb2 的每 4 位 buf[4],对应的关系是 buf[0]=’00|b[0]高六位“；buf[1]=“b[0]低 4 位 | b[1]高 4 位”；buf[2]=“b[1]低 6 位 | b3 高 2 位”； buf[3]=”00 | b[2]低 6 位“。

```
}
for (int i = 0; i < bytes.length; i += 3) {
    byte b = bytes[i];
    byte b2 = bytes[i + 1];
    byte b3 = bytes[i + 2];
    sb.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt((b >> 2) & 63));
    sb.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt(((b << 4) | (b2 >> 4)) & 63));
    sb.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt(((b2 << 2) | (b3 >> 6)) & 63));
    sb.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt(b3 & 63));
}
```

D) 根据上述逻辑分析，直接写代码解密即可得到正确答案：

```
public class Lab9 {

    public static void main(String[] args) {
        StringBuffer flag=new StringBuffer();
        String src="TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV";
        String result="Zt/aZPBpn5J2vymPf3Yun7v6d7ruflnBn5DOfhYOfN6pnuWMX5TYwNvL";
        int[] buf=new int[4];
        byte[] resultbuf=new byte[3];
        for (int i = 0; i < result.length(); i+=4) {
            buf[0]= src.indexOf(result.charAt(i));
            buf[1]= src.indexOf(result.charAt(i+1));
            buf[2]= src.indexOf(result.charAt(i+2));
            buf[3]= src.indexOf(result.charAt(i+3));

            resultbuf[0]=(byte) ((buf[0]<<2)+((buf[1]<<2)>>6));
            resultbuf[1]=(byte) (((buf[2]<<2)>>4)+(buf[1]<<4));
            resultbuf[2]=(byte) (buf[3]+(buf[2]<<6));
            for (int j = 0; j < 3; j++) {
                flag.append((char) resultbuf[j]);
            }
        }
        System.out.println(flag);
    }
```

```
un: Lab9
"D:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
FLAG{bb1f4b7e-3a3a-43ef-b70d-0d8bc5520483}
```