

Lab9 Unpacking

STU ID: 20307130044

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.

先反编译，能看到两个函数，ProxyApplication 和 ReflInvoke，后者是 Java 反射调用的方法。

```

v com
  v android
    v dexshell
      > BuildConfig
      > ProxyApplication
      > R
      > ReflInvoke
    > google
```

还有一个 attachBaseContext () 函数，这个函数把壳 dex 中包含的源 apk 释放出来。把释放的 apk 进行解密。把源 apk 中的 lib 目录中的文件复制到当前程序(壳)的路径下。

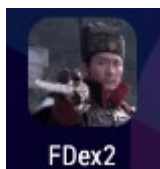
```
protected void attachBaseContext(Context base) {
    super.attachBaseContext(base);
    File odex = this.getDir("payload_odex", 0);
    File libs = this.getDir("payload_lib", 0);
    this.mDexAbsolutePath = odex.getAbsolutePath();
    this.mLibAbsolutePath = libs.getAbsolutePath();
    this.mSrcApkAbsolutePath = odex.getAbsolutePath() + "/payload.apk";
    File srcApkFile = new File(this.mSrcApkAbsolutePath);
    if(!srcApkFile.exists()) {
        try {
            srcApkFile.createNewFile();
        }
        catch(IOException e) {
            e.printStackTrace();
            this.releaseSrcApkAndSrcLibFiles(this.getDexFileFromShellApk());
            goto label_34;
        }
    }

    this.releaseSrcApkAndSrcLibFiles(this.getDexFileFromShellApk());
}
```

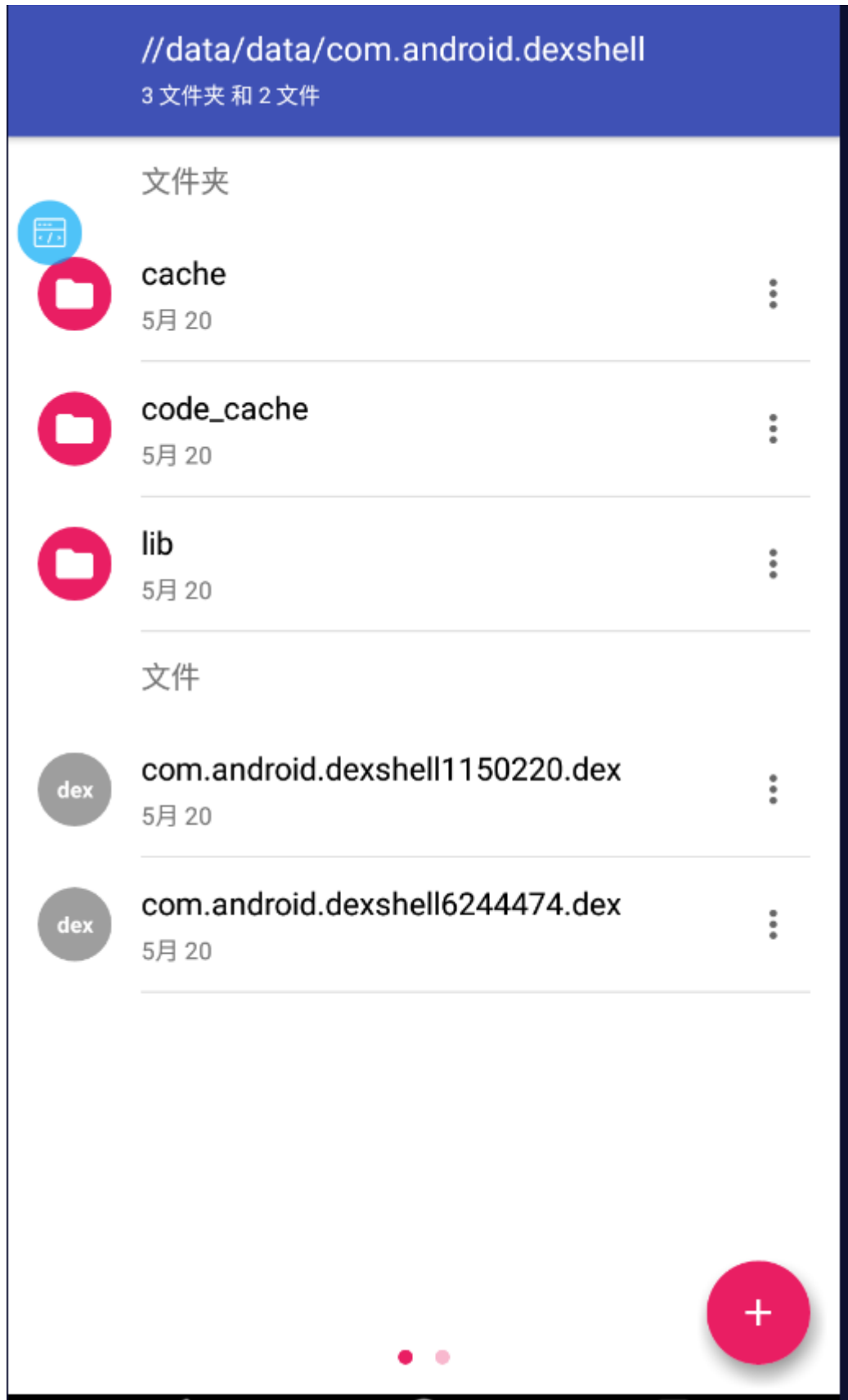
这么多特征，大概就只是 dex 壳，dex 壳是比较基础的壳，只是将源 APK 加密后放入 dex 文件中，在运行时进行释放

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















这次我就使用了专门解这种简单壳的 FDex2，这个能直接抓到中途重新释放的源 dex（需要 xposed 模块）



下面是它解析出来的 dex



然后用一个 dex2jar 软件就自动将 dex 可执行代码转为 jar 包

	com.android.dexshell1150220.dex	2022/5/20 16:21	DEX 文件	1,124 KB	
	com.android.dexshell1150220-dex2j...	2022/5/20 16:37	Executable Jar File	1,319 KB	←
	com.android.dexshell1150220-error....	2022/5/20 16:37	ZIP 文件	36 KB	
	com.android.dexshell6244474.dex	2022/5/20 16:21	DEX 文件	6,099 KB	
	com.android.dexshell6244474-dex2j...	2022/5/20 16:38	Executable Jar File	2,811 KB	←
	com.android.dexshell6244474-error....	2022/5/20 16:38	ZIP 文件	7 KB	
	d2j_invoke.bat	2014/10/27 17:32	Windows 批处理...	1 KB	
	d2j_invoke.sh	2014/10/27 17:32	Shell Script	2 KB	
	d2j-baksmali.bat	2014/10/27 17:32	Windows 批处理...	1 KB	
	d2j-baksmali.sh	2014/10/27 17:32	Shell Script	2 KB	
	d2j-dex2jar.bat	2014/10/27 17:32	Windows 批处理...	1 KB	
	d2j-dex2jar.sh	2014/10/27 17:32	Shell Script	2 KB	←
	d2j-dex2smali.bat	2014/10/27 17:32	Windows 批处理...	1 KB	
	d2j-dex2smali.sh	2014/10/27 17:32	Shell Script	2 KB	

然后再用 jeb 反编译这两个 jar 包，就可以看到反编译的代码了

```
public class MainActivity.a implements View.OnClickListener {
    public final TextView D;
    public final MainActivity c;

    public MainActivity.a(MainActivity arg1, TextView arg2) {
        this.c = arg1;
        this.D = arg2;
        super();
    }

    @Override // android.view.View.OnClickListener
    public void onClick(View arg11) {
        String v1_1;
        Context v0_2;
        byte[] v0 = this.D.getText().toString().getBytes();
        StringBuilder v4 = new StringBuilder();
        int v2 = v0.length % 3;
        String v1 = "";
        if(v2 > 0) {
            v0 = Arrays.copyOf(v0, v0.length + 3 - v2);
            while(v2 < 3) {
                v1 = d.c(v1, "-");
                ++v2;
            }
        }

        int v2_1;
        for(v2_1 = 0; v2_1 < v0.length; v2_1 += 3) {
            int v5 = v0[v2_1];
            int v6 = v0[v2_1 + 1];
            int v7 = v0[v2_1 + 2];
            v4.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt(v5 >> 2 & 0x3F));
            v4.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt((v5 << 4 | v6 >> 4) & 0x3F));
            v4.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt((v6 << 2 | v7 >> 6) & 0x3F));
            v4.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt(v7 & 0x3F));
        }
    }
}
```

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

```
public void onClick(View arg11) {
    String v1_1;
    Context v0_2;
    byte[] v0 = this.D.getText().toString().getBytes();
    StringBuilder v4 = new StringBuilder();
    int v2 = v0.length % 3;
    String v1 = "";
    if(v2 > 0) {
        v0 = Arrays.copyOf(v0, v0.length + 3 - v2);
        while(v2 < 3) {
            v1 = d.c(v1, "-");
            ++v2;
        }
    }

    int v2_1;
    for(v2_1 = 0; v2_1 < v0.length; v2_1 += 3) {
        int v5 = v0[v2_1];
        int v6 = v0[v2_1 + 1];
        int v7 = v0[v2_1 + 2];
        v4.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt(v5 >> 2 & 0x3F));
        v4.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt((v5 << 4 | v6 >> 4) & 0x3F));
        v4.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt((v6 << 2 | v7 >> 6) & 0x3F));
        v4.append("TahNbJyRm1edXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt(v7 & 0x3F));
    }
}
```

核心代码的第一部分是一个简单的模 3 补足，用“-”补足，d.c 的代码找的就是简单的字符串相加，贴在下面了

```

package b;

import androidx.recyclerview.widget.RecyclerView;
import org.xmlpull.v1.XmlPullParser;

public final class d {
    public static float a(float arg1, float arg2, float arg3, float arg4) {
        return (arg1 - arg2) * arg3 + arg4;
    }

    public static String b(RecyclerView arg1, StringBuilder arg2) {
        arg2.append(arg1.z());
        return arg2.toString();
    }

    public static String c(String arg1, String arg2) {
        return arg1 + arg2;
    }

    public static String d(String arg1, String arg2, String arg3) {
        return arg1 + arg2 + arg3;
    }
}

```

然后就是核心部分：

```

int v2_1;
for(v2_1 = 0; v2_1 < v0.length; v2_1 += 3) {
    int v5 = v0[v2_1];
    int v6 = v0[v2_1 + 1];
    int v7 = v0[v2_1 + 2];
    v4.append("TahNbJyRm1edXvwirZ37WGl8nfQUdSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt(v5 >> 2 & 0x3F));
    v4.append("TahNbJyRm1edXvwirZ37WGl8nfQUdSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt((v5 << 4 | v6 >> 4) & 0x3F));
    v4.append("TahNbJyRm1edXvwirZ37WGl8nfQUdSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt((v6 << 2 | v7 >> 6) & 0x3F));
    v4.append("TahNbJyRm1edXvwirZ37WGl8nfQUdSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV".charAt(v7 & 0x3F));
}

String v0_1 = v4.toString();
if(v0_1.substring(0, v0_1.length() - v1.length()) + v1.equals("Zt/aZPBpn5J2vymPf3Yun7v6d7ruf1nBn5D0fhY0fN6pnuWMX5TYwNvL")) {
    v0_2 = this.c(getApplicationContext());
    v1_1 = "RIGHT!";
}

```

这里将每 3 个字符转 ascii 后，通过一字符串定位加密成另一串，如果相等就正确了，主要看这 3 个字符到 4 个字符的变化：

与 0x3f 相与说明每次数据只要后六个二进制位，第一个是 v5 的高 6 位，第二个是 v5 的低 2 位 + v6 的高 4 位，第三个是 v6 的低 4 位 + v7 的高 2 位，第四个是 v7 的低 6 位，所以逆向就是把这几位拼起来就对了

```

1
2 result = "Zt/aZPBpn5J2vymPf3Yun7v6d7ruf1nBn5DOfhYOfN6pnuWMMXSTYwNvL"
3 string = "TahNbJyRmiedXvwirZ37WGl8nfQUDSF+x6p5tg2s094EoBzIO/AuYMKPHckjCLqV"
4 origin = ""
5 for i in range(0, len(result), 4):
6     position = string.find(result[i + 3])
7     v7_low6 = position
8     position = string.find(result[i + 2])
9     v7_high2 = position % 4
10    v7 = (v7_high2 << 6) + v7_low6
11    v6_low4 = position >> 2
12    position = string.find(result[i + 1])
13    v6_high4 = position % 16
14    v6 = (v6_high4 << 4) + v6_low4
15    v5_low2 = position >> 4
16    position = string.find(result[i])
17    v5_high6 = position
18    v5 = ((position << 2) + v5_low2
19    print((v5,v6,v7))
20    origin = origin + chr(v5) + chr(v6) + chr(v7)
21 print(origin)
22

```

问题 输出 终端 调试控制台

```

(98, 49, 102)
(52, 98, 55)
(101, 45, 51)
(97, 51, 97)
(45, 52, 51)
(101, 102, 45)
(98, 55, 48)
(100, 45, 48)
(100, 56, 98)
(99, 53, 53)
(50, 48, 52)
(56, 51, 125)
FLAG{bb1f4b7e-3a3a-43ef-b70d-0d8bc5520483}

```



f4b7e-3a3a-43ef-b70d-0d8bc5520483}

CHECK



RIGHT!



