Lab9 Unpacking

STU ID: 20307130001
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.**

2. 解包发现，初始有一个大的包，加载时会再加载一遍所有的包，故有两个大的包和一个小的包。
3. 上面的代码使用了 DEX 加壳技术。具体来说，这段代码在 attachBaseContext 方法中对应用进行了 DEX 加壳处理。
4. 在 attachBaseContext 方法中，首先获取了应用的私有目录，然后将 payload.dex 文件保存到了私有目录下的 payload_odex 子目录中。这个 payload.dex 文件就是加壳后的 DEX 文件。
5. 接着，使用 DexClassLoader 类加载器加载了 payload.dex 文件，用于替代原始的 ClassLoader，还会删除原始的 odex 目录和 lib 目录。
6. mDexAbsolutePath 和 mSrcApkAbsolutePath: 这两个变量分别保存了应用的私有目录中的 payload_odex 子目录和 payload.apk 文件的绝对路径，用于替代原始的 DEX 文件。
7. getDexFileFromShellApk()方法：这是一个本地方法，用于获取加壳后的 payload.dex 文件的内容。
8. DexClassLoader 类加载器的使用：在 attachBaseContext 方法中，使用 DexClassLoader 加载器加载了 payload.dex 文件，用于替代原始的 ClassLoader。这样在运行时，应用会使用经过加壳处理的 DEX 文件。
9. recursiveDeleteFile 方法：递归地删除应用私有目录下的 payload_odex 子目录和 payload_lib 子目录。
10. 通过这种方式，原始的 DEX 文件被加密或重组后，应用在运行时只使用经过加壳处理的 DEX 文件。

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

我通过 frida-dexdump 获得的。

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

我直接看脱壳后的代码，由于如果输入的不是 3 的倍数那么 v3 就会存在'-'，而所需要的字符串中没有'-'，所以 v3 是 3 的倍数长度，且长度为 57。

转换后为 Zt/aZPBpn5J2vymPf3Yun7v6d7ruflnBn5DOfhYOfN6pnuWMX5TYwNvL，对应的索引为 17，36，49，1，17，55，45，34，24，35，5，38，13，6，8，55，25，18，52，51，24，19，13，33，11，19，16，51，25，22，24，45，24，35，28，48，25，2，52，48，25，3，33，34，24，51，20，53，12，35，0，52，14，3，13，61，故原文为 FLAG{bb1f4b7e-3a3a-43ef-b70d-0d8bc5520483}。

解析用的代码为：

```
#include<stdio.h>
#include<string.h>
int main(){
	int
a[]={17,36,49,1,17,55,45,34,24,35,5,38,13,6,8,55,25,18,52,51,24,19,13,33,11,19,16,51,25,22,24,45,24,35,28,48,25,2,52,48,25,3,33,34,24,51,20,53,12,35,0,52,14,3,13,61};
	char v4;
	int aa,b,c,d,x,y,z;
	for(int i=0;i<sizeof(a)/sizeof(int);i+=4){
		aa=(a[i]);
		b=(a[i+1]);
		c=(a[i+2]);
		d=(a[i+3]);
		x=4*aa+b/16;
		y=(b%16)*16+(c/4);
		z=(c%4)*64+d;
		printf("%c%c%c",x,y,z);
	}
	return 0;
}
```