爱加密脱壳实战

title: 【天翼杯安卓题二】 爱加密脱壳实战
tags:

- 安卓脱壳
  categories:
- 安卓安全
- ''
  author: hac425
  date: 2017-10-22 19:33:00

前言

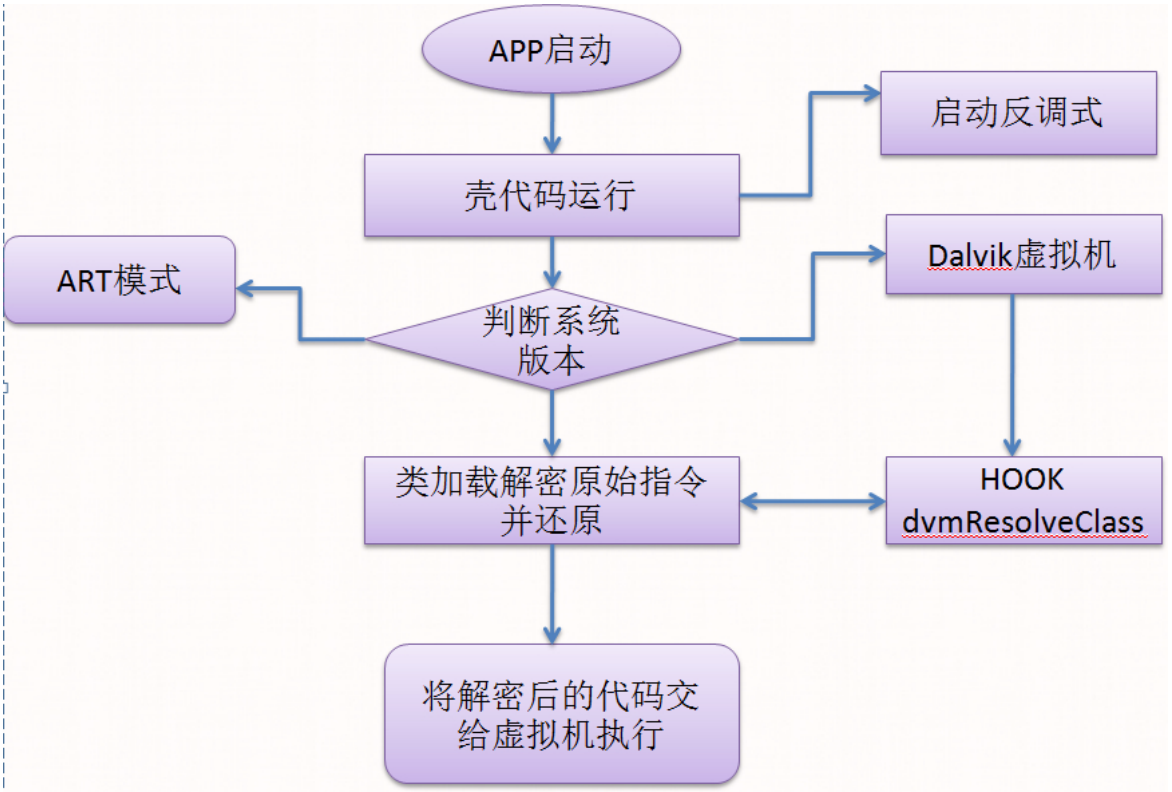
这个apk使用爱加密加密，加密时间是2017.6月。这个题其实就是个脱壳题，脱完立马见flag。(出题人也太懒了)

题目链接：https://gitee.com/hac425/blog_data/blob/master/app02.apk

壳介绍

爱加密的壳16年年底就已经开始通过 `hook dvmResolveClass` ，在调用具体方法时解密方法指令，然后将 DexFile结构体中的对应方法的 `md->insns` 指向 解密后的方法指令数据区，然后进入 `真正的dvmResolveClass`中执行指令，执行完后在重新加密指令，这样就可以防止 `dexhunter` 等工具在内存中 `dump dex` 文件。

流程图



图片来源

脱壳

由上面可以知道，在`dvmResolveClass`函数执行的时候，代码是已经还原好了的。这时我们去`dump`相应的指令就是正确的指令。于是修改 `dvmResolveClass` 的代码，`dump`方法的数据。
修改 `dvmResolveClass` 函数:

```
/*  add dump .....*/

  char key_str[20] = "jiajiatest";
  int fd=open("/data/local/tmp/resolve_class_config",O_RDONLY,0666);
  if(fd!=-1){
    int len = read(fd,key_str,19);
    key_str[len-1] = '\x00';
    key_str[len] = '\x00';
```

```
    close(fd);
  }
  ALOGI("The key_str ---> %s----referrer->descriptor--->%s--", key_str, referrer->descriptor);

if(strstr(referrer->descriptor, key_str)){
       char task_name[] = "task_name";
      char *logbuf = new char[1024];
      char path[50] = {0};
      sprintf(path, "/data/local/tmp/%s_dump_%d", key_str, getpid());
      FILE *fpw = fopen(path, "awb+");
      for(int i=0; i < referrer->directMethodCount; i++){
        Method* md = &referrer->directMethods[i];
        const char* mName_d = md->name;
        const u2 insSize_d = md->insSize;
        const u2* insns_d = md->insns;
        const u2 methodIdx_d = md->methodIndex;
        u4 insns_d_size = dvmGetMethodInsnsSize(md);
// ALOGI("hacklh_md---->%p, i-->%d, directMethodCount-->%d", md, i,referrer->directMethodCount);
        sprintf(logbuf,"-------------- (KL)resolving [class=%s, method=%s, methodIndex=%u, insSize=%u, insns_d=%x, codeSize=%d] in pid: %d(name: %s)",referrer->descriptor,mName_
        LOGD("%s",logbuf);
        if(fpw != NULL){
          fwrite(logbuf,1,strlen(logbuf),fpw);
          fflush(fpw);
          fwrite((u1*)insns_d,1,insns_d_size*2, fpw);
          fflush(fpw);
        }else{
          LOGD("----(KL)open %s fail!", path);
        }
      }
      for(int i=0; i < referrer->virtualMethodCount; i++){
        Method* mv = &referrer->virtualMethods[i];
        const char* mName_v = mv->name;
        const u2 insSize_v = mv->insSize;
        const u2* insns_v = mv->insns;
        const u2 methodIdx_v = mv->methodIndex;
        u4 insns_v_size = dvmGetMethodInsnsSize(mv);
        sprintf(logbuf,"-------------- (KL)resolving [class=%s, method=%s, methodIndex=%u, insSize=%u, insns_d=%x, codeSize=%d] in pid: %d(name: %s)",referrer->descriptor,mName_
        LOGD("%s",logbuf);
        if(fpw != NULL){
          fwrite(logbuf,1,strlen(logbuf),fpw);
          fflush(fpw);
          fwrite((u1*)insns_v,1,insns_v_size*2, fpw);
          fflush(fpw);
        }else{
          LOGD("%s","----(KL)open file fail!");
        }
      }
      if(fpw != NULL){
        fclose(fpw);
      }
      delete logbuf;
/*  add end .....*/
```

dump 之后我们需要把指令 patch 到 dex 对应位置上去。

patch 的方式有很多种，我选择使用 ida 脚本对他进行 patch。我觉得ida 就是一个各种文件格式的 loader，我们可以在 ida 中修改文件的内容，然后可以让 ida把修改应用到文件中，以完成 patch。 因此在 IDA中 patch 代码十分的方便，而且也很方便的查看 patch 后的结果。patch 代码的流程是：

读取 dump 的方法指令 ---> 定位相应方法指令数据区在 ida 中的位置----> patch

代码如下：

```
#! /usr/bin/python
# -*- coding: utf8 -*-

# 该脚本用于在ida中使用dump下来的method指令对 dex 进行Patch
import re
from dex_parser import dex

#存储 存放dump数据的字典
```

```
data_array = []
#用来避免多次patch
patched = []
file_data = ""

def parse_meta_data(data=""):
    # print data
    ret = {}
    tokens = re.findall("\[class=(.*?),.*?method=(.*?),.*?codeSize=(.*?)\]",data)
    # print tokens

    ret['class_name'] = tokens[0][0][1:].replace('/','.').replace(';','')
    ret['method'] = tokens[0][1]
    ret['code_size'] = int(tokens[0][2]) * 2 #dex文件格式定义，总大小为 codeSize*2
    # print ret
    return ret

#注释，用于给ida执行
# def patch_byte(a, b):
#     print hex(b),

def patch(dest, src, size):
    print "dest::{}, src::{}, size::{}".format(dest, src, size)
    for i in range(size):
        patch_byte(dest + i, int(file_data[ src + i].encode('hex'), 16))

    print "\n"

def parse_dump_data(filename):
    global file_data
    with open(filename, "rb") as fp:
        file_data = fp.read()

    #使用正则表达式把说明dump数据的元数据加载到内存
    all_item = re.findall("-------------- \(KL\)resolving(.*?) in pid:.*?\(name: task_name\)", file_data)
    offset = 0
    for meta_data in all_item:
        try:
            #使用字典组织数据
            #{'class_name': 'com.example.jiajiatest.MainActivity', 'code_size': 306, 'method': 'add', 'data_offset': 7175}

            ret = parse_meta_data(meta_data)
            data_addr = file_data.find('(name: task_name)', offset) + 17
            ret['data_offset'] = data_addr
            data_array.append(ret)
            offset = data_addr
        except Exception as e:
            raise e

    return data_array

def get_method_addr(method_data, signature_str):
    for md_name in method_data:
        if signature_str in md_name:
            return method_data[md_name]
    return -1

def patch_dex(dump_data_file, dex_file):
    dump_data = parse_dump_data(dump_data_file)
    dex_obj = dex.dex_parser(dex_file)
    method_data = dex_obj.get_class_data()

    for item in dump_data:
        signature_str = "{}::{}".format(item['class_name'], item['method'])
        if  signature_str not in patched:

            #获取要patch的目标地址
            addr = get_method_addr(method_data, signature_str)
```

```
            if addr == -1:
                print "{} can't get insns addr".format(signature_str)
                continue
            #do patch
            print "patch " + signature_str,
            patch(addr,item['data_offset'],item['code_size'])
            patched.append(signature_str)

    # print patched
    # for i in patched:
    #     print i

import pprint

patch_dex("F:\code_workplace\ida_script\jiajiatest_dump_20406","F:\code_workplace\ida_script\classes.dex" )
if __name__ == '__main__':
    print "comming main"
    # parse_dump_data("F:\code_workplace\ida_script\jiajiatest_dump_20406")
    # patch_dex("F:\code_workplace\ida_script\jiajiatest_dump_20406","F:\code_workplace\ida_script\classes.dex" )
    # dex_obj = dex.dex_parser("F:\code_workplace\ida_script\classes.dex")
    # class_data = dex_obj.get_class_data()
    # pprint.pprint(class_data)



    # parse_meta_data("-------------- (KL)resolving [class=Lcom/example/jiajiatest/HttpRunner;, method=makeImgHttpGET, methodIndex=13, insSize=2, insns_d=6daf04d8, codeSize=270]
```

patch**前后对比**:

patch前

```
# Source file: HttpRunner.java
public java.lang.String com.example.jiajiatest.HttpRunner.makeHttpGET(
      java.lang.String p0)
this = v11
p0 = v12
const/4                         v0, 0

locret:
return-object                   v0
```

```
# catch Exception:

loc_77ECE:
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
```

patch后

```
DDC p0 = v12
DDC HttpRunner_makeHttpGET@LL:.byte 0x1A
DDC # ---------------------------------------------------------------------------
DDE locret:
DDE                 .byte    0
DDF                 .byte    0
DE0 # ---------------------------------------------------------------------------
DE0                 sput-object                 v8, HttpRunner_jsonString
DE4                 const-string                v8, aHttp # "http"
DE8                 invoke-static               {v8, p0}, <int Log.d(ref, ref) imp. @ _def_Log_d@ILL>
DEE # try 0x77DEE-0x77E96:
DEE
DEE loc_77DEE:                              # DATA XREF: CODE:00078020↓i
DEE                 new-instance                v1, <t: DefaultHttpClient>
DF2                 invoke-direct               {v1}, <void DefaultHttpClient.<init>() imp. @ _def_DefaultHttpClient__init_@V>
DF8                 invoke-virtual              {v1}, <ref DefaultHttpClient.getParams() imp. @ _def_DefaultHttpClient_getParams@L>
DFE                 move-result-object          v8
E00                 const-string                v9, aHttpConnection # "http.connection.timeout"
E04                 const                       v10, 0xEA60
E0A                 invoke-static               {v10}, <ref Integer.valueOf(int) imp. @ _def_Integer_valueOf@LI>
E10                 move-result-object          v10
E12                 invoke-interface            {v8, v9, v10}, <ref HttpParams.setParameter(ref, ref) imp. @ _def_HttpParams_setParameter@L
E18                 new-instance                v3, <t: HttpGet>
E1C                 invoke-direct               {v3, p0}, <void HttpGet.<init>(ref) imp. @ _def_HttpGet__init_@VL>
E22                 invoke-virtual              {v1, v3}, <ref DefaultHttpClient.execute(ref) imp. @  def DefaultHttpClient_execute@LL>
```

这时已经可以看到程序的主体逻辑了。然后查看字符串就可以拿到flag.....

我干的傻事

- 代码循环条件忘记写了，导致越界，一打开应用就报错。

- 文件打开失败，貌似是权限问题，我直接暴力把 `/data/local/tmp` 改成 777

总结

分析安卓底层代码的错误，要关注 `logcat` 日志，找到出问题的代码点，然后把库的带符号版本放到 `ida` 中分析分析bug, 要看代码的关键逻辑， 判断条件等。

最后

要多实践，如有问题请在下面评论。

来源： https://www.cnblogs.com/hac425/p/9416964.html