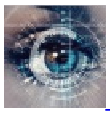
cocos2dx-Lua引擎游戏脚本及图片资源解密与DUMP

原创

asmcvc 于 2017-01-05 21:11:56 发布 13682 收藏 8
分类专栏： Android安全 NDK 文章标签： cocos2dx 游戏 解密 少年三国志 全民水浒

Android安全 同时被 2 个专栏收录
35 篇文章 5 订阅
订阅专栏

NDK
9 篇文章 0 订阅
订阅专栏

分析目标

- 少年三国志，包名：com.youzu.android.snsgz
- 全民水浒，包名：com.tencent.Q108

下面分析的主要是少年三国志。

Lua脚本解密与DUMP

LuaJit IDA分析调用树：

> **AppDelegate::applicationDidFinishLaunching**(AppDelegate *__hidden this) EXPORT _ZN11AppDelegate29applicationDidFinishLaunchingEv
>
> **cocos2d::CCLuaEngine::defaultEngine**(cocos2d::CCLuaEngine *__hidden this) EXPORT _ZN7cocos2d11CCLuaEngine13defaultEngineEv
>
> **cocos2d::CCLuaEngine::init**(cocos2d::CCLuaEngine *__hidden this) EXPORT _ZN7cocos2d11CCLuaEngine4initEv
>
> **cocos2d::CCLuaStack::create**(cocos2d::CCLuaStack *__hidden this) EXPORT _ZN7cocos2d10CCLuaStack6createEv
>
> **cocos2d::CCLuaStack::init**(cocos2d::CCLuaStack *__hidden this) EXPORT _ZN7cocos2d10CCLuaStack4initEv
>
> **cocos2dx_lua_loader**
>
> **cocos2d::CCLuaStack::lua_loadbuffer**(lua_State , *char const*, int, char const*) EXPORT *ZN7cocos2d10CCLuaStack14lua_loadbufferEP9lua_StatePKciS4*

cocos2d::CCLuaStack::lua_loadbuffer先调用以下函数解密： **cocos2d::extra::CCCrypto::decryptUF**(uchar ,*int,int* ,int *)
EXPORT *ZN7cocos2d5extra8CCCrypto9decryptUFEPhiPiS3*
最后再调用：**luaL_loadbuffer**

因此可以直接对**luaL_loadbuffer**进行HOOK，进而DUMP出Lua脚本，网上搜索函数声明：

```
int luaL_loadbuffer (lua_State *L, const char *buff, size_t sz, const char *name);
```

进而实现HOOK代码：

```
//orig function copy
int (*luaL_loadbuffer_orig)(void *L, const char *buff, int size, const char *name) = NULL;

//local function
int luaL_loadbuffer_mod(void *L, const char *buff, int size, const char *name) {
    LOGD("[dumplua] luaL_loadbuffer name: %s lua: %s", name, buff);
    return luaL_loadbuffer_orig(L, buff, size, name);
}

void hook() {
    LOGD("[dumplua] hook begin");
    void *handle = dlopen("libgame.so", RTLD_NOW);
    if (handle == NULL) {
        LOGE("[dumplua]dlopen err: %s.", dlerror());
        return;
    }else{
        LOGD("[dumplua] libgame.so dlopen OK!");
    }


    void *pluaL_loadbuffer = dlsym(handle, "luaL_loadbuffer");
    if (pluaL_loadbuffer == NULL){
        LOGE("[dumplua] lua_loadbuffer not found!");
        LOGE("[dumplua] dlsym err: %s.", dlerror());
    }else{
        LOGD("[dumplua] luaL_loadbuffer found!");
        MSHookFunction(pluaL_loadbuffer, (void *)&luaL_loadbuffer_mod, (void **)&luaL_loadbuffer_orig);
    }
}
```

运行后拦截到的输出信息：

```
01-05 19:29:27.674 13191-13215/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: assets/scripts/main.l
    function __G__TRACKBACK__(errorMessage)

      print("----------------------------------------")
      print("LUA ERROR: " .. tostring(errorMessage) .. "\n")
      local traceback = debug.traceback("", 2)
      print(traceback)
      print("----------------------------------------")


      --只有G_Report初始过后才会对错误日志做处理
      if G_Report ~= nil then
          G_Report:onTrackBack(errorMessage, traceback)
      end


      if SHOW_EXCEPTION_TIP and uf_notifyLayer ~= nil then
        uf_notifyLayer:getDebugNode():removeChildByTag(10000)
        local text = tostring(errorMessage)
          require("upgrade.ErrMsgBox").showErrorMsgBox(text)

      end
    end



    function traceMem(desc)
      if desc == nil then
          desc = "memory:"
      end


      if CCLuaObjcBridge then
          local callStaticMethod = CCLuaObjcBridge.callStaticMethod

          local ok, ret = callStaticMethod("NativeProxy", "getUsedMemory", nil)

          if ok then
              pri
01-05 19:29:27.679 13191-13215/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: upgrade.AntiAddiction
01-05 19:29:27.679 13191-13215/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: upgrade.ComSdkUtils l
01-05 19:29:27.684 13191-13215/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: upgrade.config lua: L
01-05 19:29:27.684 13191-13215/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: upgrade.ConfigLayer l
01-05 19:29:27.684 13191-13215/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: upgrade.EffectNode_Up
01-05 19:29:27.684 13191-13215/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: upgrade.ErrMsgBox lua
01-05 19:29:27.684 13191-13215/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: upgrade.game lua: LJ�
01-05 19:29:27.684 13191-13215/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: upgrade.NativeCallUti
01-05 19:29:27.684 13191-13215/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: upgrade.NativeProxy l
01-05 19:29:27.684 13191-13215/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: upgrade.Patcher lua:
01-05 19:29:27.689 13191-13215/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: upgrade.SplashLayer l
01-05 19:29:27.689 13191-13215/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: upgrade.upgrade lua:
```

可见，有些Lua脚本是源码形式，有些是LuaJit编译的，可以改写以上代码把脚本DUMP到文件中再进一步分析，此处略。

资源解密与DUMP

主要函数： cocos2d::CCImage::**initWithImageFile**调用 cocos2d::CCImage::**initWithImageData**

但是IDA分析发现**initWithImageData**会调用**cocos2d::extra::CCCrypto::decryptXXTEA**和**cocos2d::extra::CCCrypto::decryptUF**进行解密，最后再加载图片资源。以下是initWithImageData部分代码：

```
      if ( s )
        {
          v12 = (unsigned __int8 *)strlen(s);
          v13 = (void *)cocos2d::extra::CCCrypto::decryptXXTEA(v9, v11, (int)s, v12, (int)v24, v22);
          v14 = v13;
          if ( v13 )
            v9 = (cocos2d::extra::CCCrypto *)v13;
          goto LABEL_28;
        }
        v14 = 0;
        if ( (signed int)a3 > 3 && *(_BYTE *)this == 85 && *((_BYTE *)this + 1) == 70 )
        {
          v25 = 0;
          cocos2d::extra::CCCrypto::decryptUF(this, (int)a3, (int)&v25, v24, v21);
          v15 = v25 >> 4;
          if ( (v25 & 0xFu) <= 9 )
            *(_DWORD *)(v8 + 36) = v25 & 0xF;
          switch ( v15 )
          {
            case 1:
              v16 = 1067030938;
              break;
            case 2:
              v16 = 1068708659;
```

也即会调用cocos2d::extra::CCCrypto::decryptXXTEA和cocos2d::extra::CCCrypto::decryptUF进行解密操作。我们看下**cocos2d::extra::CCCrypto::decryptUF**这个函数，通过IDA的F5插件，并不断修改变量名可以获得一个比较清晰的C代码。

```
      int __fastcall cocos2d::extra::CCCrypto::decryptUF(cocos2d::extra::CCCrypto *pInBuff, int nlen, int a3,
      {
        cocos2d::extra::CCCrypto *pInBuff2; // r5@1
        int *pOutLen2; // r7@1
        int v7; // r3@4
        int v8; // r6@5
        int v9; // r1@6
        int result; // r0@9
        int v11; // r4@10
        int v12; // r6@12
        int v13; // r6@15
        signed int v14; // r0@19
        int v15; // [sp+0h] [bp-28h]@5
        int v16; // [sp+0h] [bp-28h]@10
        int v17; // [sp+4h] [bp-24h]@5

        pInBuff2 = pInBuff;
        pOutLen2 = pOutLen;
        if ( nlen <= 3 )
        {
          v14 = 1;
          return -v14;
        }
        if ( *(_BYTE *)pInBuff != 'U' || *((_BYTE *)pInBuff + 1) != 'F' )
        {
```

```
  {
    v14 = 2;
    return -v14;
  }
  *(_DWORD *)a3 = *((_BYTE *)pInBuff + 2);
  v7 = *((_BYTE *)pInBuff + 3);
  if ( v7 == 1 )
  {
    v15 = nlen - 5;
    v17 = *((_BYTE *)pInBuff + 4);
    v8 = 0;
    while ( v8 < v15 )
    {
      v9 = (v8++ + v17) % 0x21;
      *(_BYTE *)pInBuff2 = *((_BYTE *)pInBuff2 + 5) ^ byte_6D192C[v9];
      pInBuff2 = (cocos2d::extra::CCCrypto *)((char *)pInBuff2 + 1);
    }
    *pOutLen2 = v15;
  }
  else
  {
    result = 0;
    if ( v7 != 2 )
      return result;
    v11 = 0;
    v16 = *((_BYTE *)pInBuff2 + 4);
    do
    {
      *((_BYTE *)pInBuff2 + v11) = *((_BYTE *)pInBuff2 + nlen + v11 - 5) ^ byte_6D192C[(v11 + v16) % 33
      ++v11;
    }
    while ( v11 != 5 );
    v12 = nlen - 10;
    if ( nlen - 10 > 95 )
      v12 = 95;
    v13 = v12 + 4;
    while ( v13 >= v11 )
    {
      *((_BYTE *)pInBuff2 + v11) ^= byte_6D192C[(v11 + v16) % 33 + 33];
      ++v11;
    }
    *pOutLen2 = nlen - 5;
  }
  return 0;
}
```

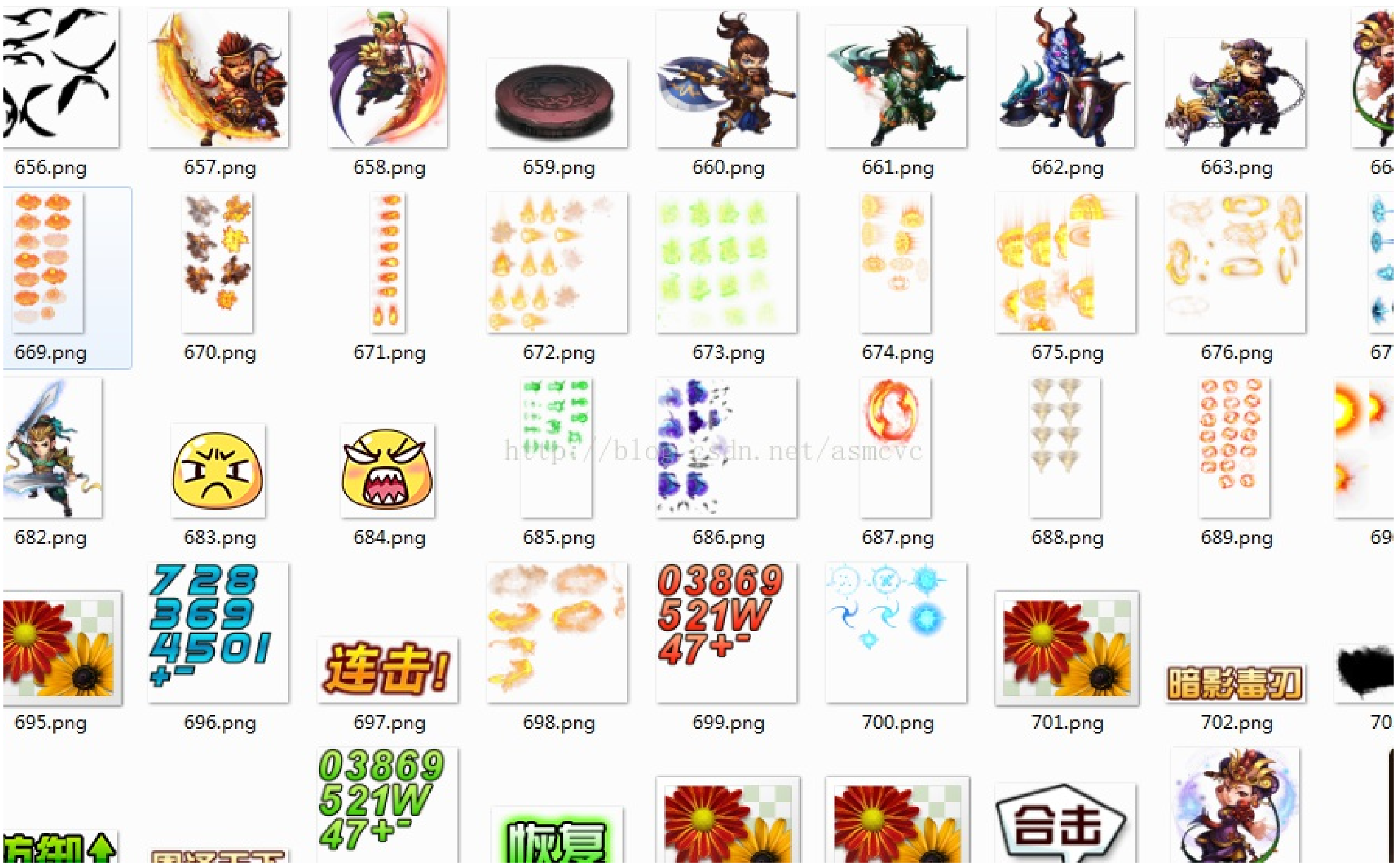
其实看到这里应该也是比较容易逆向分析出解密的算法的，应该说比较简单，可以直接写一个脚本来解密assets里的资源。但是为了保证通用性，还是写HOOK代码比较好。

本来分析以为最终都会调用_initWithWebpData、_initWithJpgData、_initWithBpgData、_initWithPngData、_initWithTiffData、_initWithRawData这些函数的，但是实际上分别HOOK后并没有被拦截，所以最后还是HOOK了下**cocos2d::extra::CCCrypto::decryptUF**。

```
static string g_strDataPath;
static int g_nCount = 1;

string getNextFilePath(const char *fileExt) {
    char buff[100] = {0};
    ++g_nCount;
    sprintf(buff, "%s/cache/%d%s", g_strDataPath.c_str(), g_nCount, fileExt);
    return buff;
}

bool saveFile(const void* addr, int len, const char *outFileName)
{
    bool bSuccess = false;
    FILE* file = fopen(outFileName, "wb+");
    if (file != NULL) {
        fwrite(addr, len, 1, file);
        fflush(file);
        fclose(file);
        bSuccess = true;
        chmod(outFileName, S_IRWXU | S_IRWXG | S_IRWXO);
    }else{
        LOGE("[%s] fopen failed, error: %s", __FUNCTION__, dlerror());
    }

    return bSuccess;
}

//hook decryptUF
int (*decryptUF_orig)(void *pInBuff, int len, int *n, int *poutlen, char *name) = NULL;
int decryptUF_mod(void *pInBuff, int len, int *n, int *poutlen, char *name) {
    int ret = decryptUF_orig(pInBuff, len, n, poutlen, name);
    saveFile(pInBuff, *poutlen, getNextFilePath(".png").c_str());
    return ret;
}

void hook() {
    //hook decryptUF
    void *decryptUF = dlsym(handle, "_ZN7cocos2d5extra8CCCrypto9decryptUFEPhiPiS3_");
    if ( decryptUF==NULL ) {
        LOGE("[dumplua] _ZN7cocos2d5extra8CCCrypto9decryptUFEPhiPiS3_ (decryptUF) not found!");
        LOGE("[dumplua] dlsym err: %s.", dlerror());
    }else{
        LOGD("[dumplua] _ZN7cocos2d5extra8CCCrypto9decryptUFEPhiPiS3_ (decryptUF) found!");
        MSHookFunction(decryptUF, (void *)&decryptUF_mod, (void **)&decryptUF_orig);
    }
}
```

我这里图方便把所有解密的数据都DUMP为/data/data/packagename/cache目录下扩展名为PNG的文件了，最后通过脚本从手机中批量提取出解密后的文件：

```
#coding:utf-8
import os

for i in range(1, 10000):
    cmd = 'adb pull /data/data/com.youzu.android.snsgz/cache/' + str(i) +'.png' + ' e:\\test'
    os.system(cmd)
```



其实通过上面的分析就可以知道，图片资源的解密和Lua的解密都是调用了相同的函数，因此解密出的文件不全是图片，还有写LuaJit的脚本文件，用十六进制编辑器打开就可以看到LJ开头的魔法数字。

全民水浒

全民水浒这个比较简单，资源直接没加密处理，解压缩APK文件就可以在assets目录下查看了。Lua脚本可以通过HOOK函数luaL_loadbuffer获得，而且可以看出只是对编译的Lua脚本做了简单的加密，可以直接DUMP出来，相对少年三国志稍微弱了一些。

```
01-05 20:04:16.569 17729-17886/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: require "UpdateScene.
01-05 20:04:16.574 17729-17886/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: UpdateScene.lua lua:
01-05 20:04:16.574 17729-17886/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: Modal.lua lua: LuaQ
01-05 20:04:16.574 17729-17886/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: UIDefine.lua lua: Lua
01-05 20:04:16.574 17729-17886/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: MessageBox.lua lua: L
01-05 20:04:16.579 17729-17886/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: SoundManager.lua lua:
01-05 20:04:16.579 17729-17886/? D/SUBSTRATEHOOK: [dumplua] luaL_loadbuffer name: SoundConfig.lua lua:
```

创作打卡挑战赛
赢取流量/现金/CSDN周边激励大奖