ndk学习9: 动态使用共享库

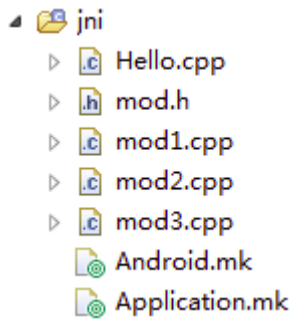
动态使用共享库函数

dll_main

环境介绍

续上节代码

目录结构:



android.mk如下:

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE    := demo

LOCAL_SRC_FILES := mod1.cpp mod2.cpp mod3.cpp

include $(BUILD_SHARED_LIBRARY)

include $(CLEAR_VARS)

LOCAL_MODULE    := Hello

LOCAL_SRC_FILES := Hello.cpp

include $(BUILD_EXECUTABLE)
```

Hello.cpp

```
#include <stdio.h>

#include <dlfcn.h>

typedef void (*FUNTYPE)();

int main(int argc, char* argv[])

{

   //加载共享库

   void *handle = dlopen("/data/local/tmp/libdemo.so", RTLD_NOW);

   if (handle == NULL)

   {

      puts(dlerror());

      return 0;
```


NDK(22)

U3D(2)

安全杂项(8)

成长

随笔档案

2017年1月 (2)

2016年12月 (1)

2016年11月 (4)

2016年9月 (8)

2016年8月 (40)

2016年7月 (16)

2016年2月 (2)

2014年11月 (1)

2014年10月 (1)

2014年9月 (4)

积分与排名

积分 - 20888

排名 - 16194

阅读排行榜

1. Android模拟位置信息(4054)

2. NDK学习三: 纯手工编译Hello World(1623)

3. ndk学习20: jni之OnLoad动态注册函数(1050)

4. Android Killer工具用法(945)

5. IDA插件栈字符串识别插件(862)

6. 各种保护机制绕过手法(835)

```
    }
    printf("handle=%p\n", handle);
    //获取导出函数
    FUNTYPE pfnMod = (FUNTYPE)dlsym(handle, "_Z4mod1v");
    if (pfnMod != NULL)
    {
        printf("address=%p\n", pfnMod);
        pfnMod();
    }
    pfnMod = (FUNTYPE)dlsym(handle, "_Z4mod2v");
    if (pfnMod != NULL)
    {
        printf("address=%p\n", pfnMod);
        pfnMod();
    }
    pfnMod = (FUNTYPE)dlsym(handle, "_Z4mod3v");
    if (pfnMod != NULL)
    {
        printf("address=%p\n", pfnMod);
        pfnMod();
    }
    pfnMod = (FUNTYPE)dlsym(handle, "mod4");
```

7. 十三. JEB破解三(784)
8. 十二. 一步步破解JEB 2.0demo版二(689)
9. U3D学习笔记1: HelloWorld(602)
10. NDK学习一: 环境搭建Eclipse篇(566)
11. 十一. 一步步破解JEB 2.0demo版一(546)
12. 源码编译绕过反调试(536)
13. IDA调试android so的.init_array数组(525)
14. NDK学习二: NDK目录结构(488)
15. ndk学习6: 使用gdb调试ndk程序一(444)

```
    if (pfnMod == NULL)
    {
        puts(dlerror());
    }
    dlclose(handle);
    return 0;
}
```

_Z4mod2v 是C++的名称粉碎

函数名可以用readelf来进行查看:

该工具在: E:\Android\android-ndk-r10b\toolchains\x86-4.6\prebuilt\windows-x86_64\bin\i686-linux-android-readelf.exe

```
E:\Code\NDK\Hello\libs\x86>readelf -s libdemo.so

Symbol table '.dynsym' contains 11 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FUNC    GLOBAL DEFAULT  UND __cxa_finalize
     2: 00000000     0 FUNC    GLOBAL DEFAULT  UND __cxa_atexit
     3: 00000000     0 FUNC    GLOBAL DEFAULT  UND __stack_chk_fail
     4: 000003d0    40 FUNC    GLOBAL DEFAULT    7 _Z4mod1v
     5: 00000000     0 FUNC    GLOBAL DEFAULT  UND puts
     6: 000003f8    40 FUNC    GLOBAL DEFAULT    7 _Z4mod2v
     7: 00000420    40 FUNC    GLOBAL DEFAULT    7 _Z4mod3v
     8: 00002014     0 NOTYPE  GLOBAL DEFAULT  ABS _edata
     9: 00002014     0 NOTYPE  GLOBAL DEFAULT  ABS __bss_start
    10: 00002014     0 NOTYPE  GLOBAL DEFAULT  ABS _end
```

类似于Windows动态调用dll的思想

dlopen打开一个so文件

dlsym根据函数名拿到函数指针

编译后使用makefile执行

在工程目录根下新建makefile:

```
MODALE_NAME := Hello
# x86 path
X86_TOOLS_PATH :=E:\Android\android-ndk-r10b\toolchains\x86-4.6\prebuilt\windows-x86_64\bin
X86_GDB_PATH := $(X86_TOOLS_PATH)\i686-linux-android-gdb.exe
X86_GDB_SERVER := E:\Android\android-ndk-r10b\prebuilt\android-x86\gdbserver\gdbserver
# arm-linux-androideabi-4.6 path
arm_tools_path :=E:\Android\android-ndk-r10b\toolchains\arm-linux-androideabi-4.6\prebuilt\windows-x86_64\bin
arm_4_6_path := $(arm_tools_path)\arm-linux-androideabi-gdb.exe
arm_gdb_server :=E:\Android\android-ndk-r10b\prebuilt\android-arm\gdbserver\gdbserver
run_arm:
    adb push .\libs\armeabi-v7a\$(MODALE_NAME) /data/local/tmp
    adb shell chmod 755 /data/local/tmp/$(MODALE_NAME)
    adb shell /data/local/tmp/$(MODALE_NAME)
run_x86:
    adb push .\libs\x86\$(MODALE_NAME) /data/local/tmp
    adb shell chmod 755 /data/local/tmp/$(MODALE_NAME)
    adb shell /data/local/tmp/$(MODALE_NAME)
```

```
run_x86_share:
    adb push .\libs\x86\$(MODALE_NAME) /data/local/tmp
    adb push .\libs\x86\libdemo.so /data/local/tmp
    adb shell chmod 755 /data/local/tmp/$(MODALE_NAME)
    adb shell /data/local/tmp/$(MODALE_NAME)
debug_x86:
    adb forward tcp:12345 tcp:12345
    adb push $(X86_GDB_SERVER) /data/local/tmp
    adb shell chmod 777 /data/local/tmp/gdbserver
    adb push .\obj\local\x86\$(MODALE_NAME) /data/local/tmp
    adb shell chmod 777 /data/local/tmp/$(MODALE_NAME)
    adb shell /data/local/tmp/gdbserver :12345  /data/local/tmp/$(MODALE_NAME)
client_x86:
    $(X86_GDB_PATH) .\obj\local\x86\$(MODALE_NAME)
# 1. target remote localhost:12345
# 2. gdb.setup
```

make run_x86_share 即可成功执行

```
zero@bingghost MINGW64 /e/Code/NDK/Hello
$ make run_x86_share
adb push .\libs\x86\Hello /data/local/tmp
[100%] /data/local/tmp/Hello
adb push .\libs\x86\libdemo.so /data/local/tmp
[100%] /data/local/tmp/libdemo.so
adb shell chmod 755 /data/local/tmp/Hello
adb shell /data/local/tmp/Hello
handle=0xb7700520
address=0xb71073d0
mod1
address=0xb71073f8
mod2
address=0xb7107420
mod3
Symbol not found:
```

mod4 会提示找不到

DLL_MAIN

修改mod1.cpp

```
#include <stdio.h>

// 初始化函数

void   _init()

{

  printf("_init\r\n");

}

// so卸载函数

void   _fini()

{
```

```
  printf("_fini\r\n");
}
// 新版本初始化函数
void __attribute__((constructor)) OnLoad()
{
  printf("OnLoad\r\n");
}
void __attribute__((destructor)) UnLoad()
{
  printf("UnLoad\r\n");
}
void __attribute__((constructor)) OnLoad2()
{
  printf("OnLoad2\r\n");
}
void __attribute__((destructor)) UnLoad2()
{
  printf("UnLoad2\r\n");
}
//隐藏函数
void __attribute__((visibility("hidden"))) mod1()
{
```

```
 printf("mod1\r\n");

}
```

运行效果:

```
zero@bingghost MINGW64 /e/Code/NDK/Hello
$ make run_x86_share
adb push .\libs\x86\Hello /data/local/tmp
[100%] /data/local/tmp/Hello
adb push .\libs\x86\libdemo.so /data/local/tmp
[100%] /data/local/tmp/libdemo.so
adb shell chmod 755 /data/local/tmp/Hello
adb shell /data/local/tmp/Hello
OnLoad
OnLoad2
handle=0xb7785520
address=0xb718c5a8
mod2
address=0xb718c5d0
mod3
Symbol not found:
UnLoad2
UnLoad
```

说明:

1. _init函数比构造函数来的早

2. hidden后在用dlsym函数无法找到

总结:

1.相关函数

dlopen()函数打开一个共享库

dlsym()函数在库中搜索一个符号

dlclose() 函数光比之前dlopen打开的库

dlerror() 函数返回一个错误消息的字符串

2.隐藏函数

Void __attribute__ ((visibility("hidden"))) fun() {}

3. so构造析构(会在so加载和卸载的时候调用)

void __attribute__ ((constructor)) Load()

void __attribute__ ((destructor)) UnLoad()

4. _init()和_fini()函数

会在so加载和卸载时调用

分类: NDK

Bingghost
关注 - 2
粉丝 - 10
+加关注

0 0

« 上一篇：ndk学习8: 编译动态库
» 下一篇：ndk学习10: linux文件系统

最新IT新闻:
· 人到中年，职场半坡
· 2017“华为系”离职员工创业榜，华为“编外”军团暗中崛起
· 2017年度Linux内核开发报告
· 黑钱好赚？研究表明，勒索软件销售额疯长
· 程序猿脱单必备指南，约会中有哪些绝不能犯的「致命硬伤」？
» 更多新闻...

最新知识库文章:
· 实用VPC虚拟私有云设计原则
· 如何阅读计算机科学类的书
· Google 及其云智慧
· 做到这一点，你也可以成为优秀的程序员
· 写给立志做码农的大学生
» 更多知识库文章...