登录

android so加固之section加密

yong夜 / 2019-06-06 08:50:00 / 浏览数 5593 安全技术 移动安全 顶(0) 踩(0)

[TOC]

引言

如何对so文件中的核心代码进行保护？

通过将核心代码写到自定义节中，并且对该节使用加密工具进行加密，在so文件执行时，利用attribute((constructor));属性，先于main执行解密函数，作用类似于java中的

实现流程

1. 确定好自定义节的名称
2. 开始加密流程
   - 遍历所有节头，根据节头名来定位需要加密的节
   - 获取节头中节的起始位置和大小，对节头指向的数据进行加密
3. 编写解密代码
   - 用属性： attribute((constructor));声明解密函数
   - 在native层编写解密函数

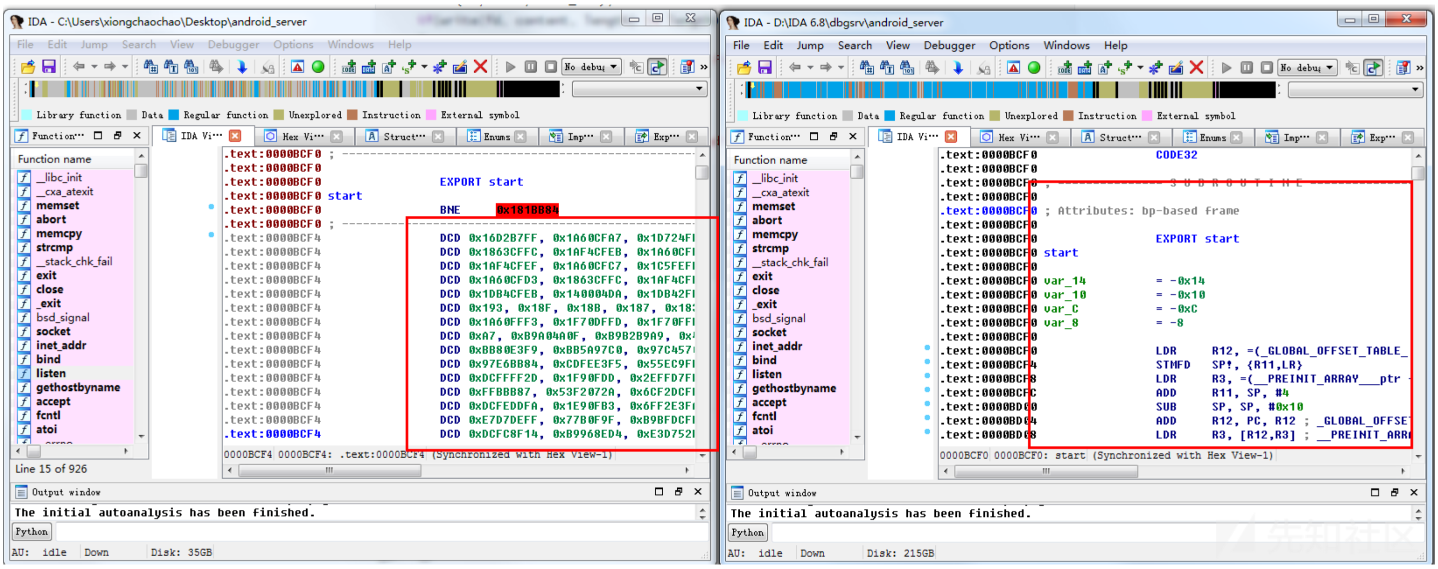
代码实现

加密流程

```
#include <stdio.h>
#include <elf.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>



int main(int argc, char** argv)
{
   int fd;
   Elf32_Ehdr ehdr;
   Elf32_Shdr shdr;
   char * section_name_table;
   int i;
   unsigned int base, length;
   char *content;


   //■■■■
   if(argc != 3)
   {
       printf("Encrypt section of elf file\n\nUsage:\n\t%s <elf_file> <section_name>\n", *argv);
       goto _error;
   }

   if((fd = open(argv[1], O_RDWR, 0777)) == -1)
   {
       perror("open");
       goto _error;
   }

   if(read(fd, &ehdr, sizeof(Elf32_Ehdr)) != sizeof(Elf32_Ehdr))
   {
       perror("read elf header");
       goto _error;
   }
```

```
//■■■■■■■■
printf("[+] Begining find section %s\n", argv[2]);
lseek(fd, ehdr.e_shoff+sizeof(Elf32_Shdr)*ehdr.e_shstrndx, SEEK_SET);
if(read(fd, &shdr, sizeof(Elf32_Shdr)) != sizeof(Elf32_Shdr))
{
    perror("read elf section header which contain string table");
    goto _error;
}

if((section_name_table = (char*) malloc(shdr.sh_size)) == NULL)
{
    perror("malloc for SHT_STRTAB");
    goto _error;
}

lseek(fd, shdr.sh_offset, SEEK_SET);
if(read(fd, section_name_table, shdr.sh_size) != shdr.sh_size)
{
    perror("read string table");
    goto _error;
}
lseek(fd, ehdr.e_shoff, SEEK_SET);
//■■■■■■■■■■■■■■■
for(i=0; i<ehdr.e_shnum; i++)
{
    if(read(fd, &shdr, sizeof(Elf32_Shdr)) != sizeof(Elf32_Shdr))
    {
        perror("read section");
        goto _error;
    }
    if(strcmp(section_name_table+shdr.sh_name, argv[2]) == 0)
    {
        base = shdr.sh_offset;
        length = shdr.sh_size;
        printf("[+] Find section %s\n", argv[2]);
        printf("[+] %s section offset is %X\n", argv[2], base);
        printf("[+] %s section size is %d\n", argv[2], length);
        break;
    }
}

//■■■■■■
lseek(fd, base, SEEK_SET);
content = (char *)malloc(length);
if(content == NULL)
{
    perror("malloc space for section");
    goto _error;
}
if(read(fd, content, length) != length)
{
    perror("read section in encrpt");
    goto _error;
}
//■■■■
for(i=0; i<length; i++)
{
    content[i] = ~content[i];
}

lseek(fd, 0, SEEK_SET);
if(write(fd, &ehdr, sizeof(Elf32_Ehdr)) != sizeof(Elf32_Ehdr))
{
    perror("write ELF header to file");
    goto _error;
}
lseek(fd, base, SEEK_SET);
if(write(fd, content, length) != length)
{
        perror("write encrypted section to file");
        goto _error;
    }

    printf("[+] Encrypt section %s completed!\n", argv[2]);
_error:
    free(section_name_table);
    free(content);
    close(fd);
    return 0;
}
```

加密前后的对比图：



IDA - C:\Users\xiongchaochao\Desktop\android_server

```
.text:0000BCF0 ; ----------------------------------------
.text:0000BCF0
.text:0000BCF0                 EXPORT start
.text:0000BCF0 start
.text:0000BCF0                 BNE     0x181BB84
.text:0000BCF0 ; ----------------------------------------
.text:0000BCF4                 DCD 0x16D2B7FF, 0x1A60CFA7, 0x1D724FI
.text:0000BCF4                 DCD 0x1863CFFC, 0x1AF4CFEB, 0x1A60CFI
.text:0000BCF4                 DCD 0x1AF4CFEF, 0x1A60CFC7, 0x1C5FEFI
.text:0000BCF4                 DCD 0x1A60CFD3, 0x1863CFFC, 0x1AF4CFI
.text:0000BCF4                 DCD 0x1DB4CFEB, 0x140004DA, 0x1DB42FI
.text:0000BCF4                 DCD 0x193, 0x18F, 0x18B, 0x187, 0x18
.text:0000BCF4                 DCD 0x1A60FFF3, 0x1F70DFFD, 0x1F70FFI
.text:0000BCF4                 DCD 0xA7, 0xB9A04A0F, 0xB9B2B9A9, 0x
.text:0000BCF4                 DCD 0xBB80E3F9, 0xBB5A97C0, 0x97C457
.text:0000BCF4                 DCD 0x97E6BB84, 0xCDFEE3F5, 0x55EC9FI
.text:0000BCF4                 DCD 0xDCFFFF2D, 0x1F90FDD, 0x2EFFD7FI
.text:0000BCF4                 DCD 0xFFBBB87, 0x53F2072A, 0x6CF2DCFI
.text:0000BCF4                 DCD 0xDCFEDDFA, 0x1E90FB3, 0x6FF2E3FI
.text:0000BCF4                 DCD 0xE7D7DEFF, 0x77B0F9F, 0xB9BFDCFI
.text:0000BCF4                 DCD 0xDCFC8F14, 0xB9968ED4, 0xE3D752I
0000BCF4 0000BCF4: .text:0000BCF4 (Synchronized with Hex View-1)
```

Line 15 of 926

The initial autoanalysis has been finished.

AU: idle   Down   Disk: 35GB

IDA - D:\IDA 6.8\dbgsrv\android_server

```
.text:0000BCF0                 CODE32
.text:0000BCF0
.text:0000BCF0 ; =============== S U B R O U T I N E ===============
.text:0000BCF0
.text:0000BCF0 ; Attributes: bp-based frame
.text:0000BCF0
.text:0000BCF0                 EXPORT start
.text:0000BCF0 start
.text:0000BCF0
.text:0000BCF0 var_14          = -0x14
.text:0000BCF0 var_10          = -0x10
.text:0000BCF0 var_C           = -0xC
.text:0000BCF0 var_8           = -8
.text:0000BCF0
.text:0000BCF0                 LDR     R12, =(_GLOBAL_OFFSET_TABLE_
.text:0000BCF4                 STMFD   SP!, {R11,LR}
.text:0000BCF8                 LDR     R3, =(__PREINIT_ARRAY___ptr -
.text:0000BCFC                 ADD     R11, SP, #4
.text:0000BD00                 SUB     SP, SP, #0x10
.text:0000BD04                 ADD     R12, PC, R12 ; _GLOBAL_OFFSE
.text:0000BD08                 LDR     R3, [R12,R3] ; __PREINIT_ARR
0000BCF0 0000BCF0: start (Synchronized with Hex View-1)
```

The initial autoanalysis has been finished.

AU: idle   Down   Disk: 215GB

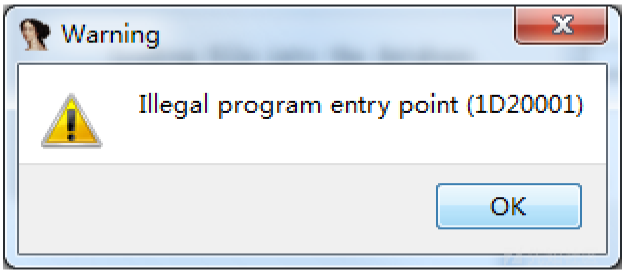
上面的加密代码只对节数据进行加密，下面我们增加几行代码，把被加密节的长度、用到的内存页数替换到文件头中的入口点和节头表偏移中去，进一步防止反汇编并且简化

那么有人会问，入口点都被填充了文件怎么执行？这里我们需要知道，对于动态链接库，e_entry
入口地址是无意义的，因为程序被加载时，设定的跳转地址是动态连接器的地址，这个字段是可以被作为数据填充的

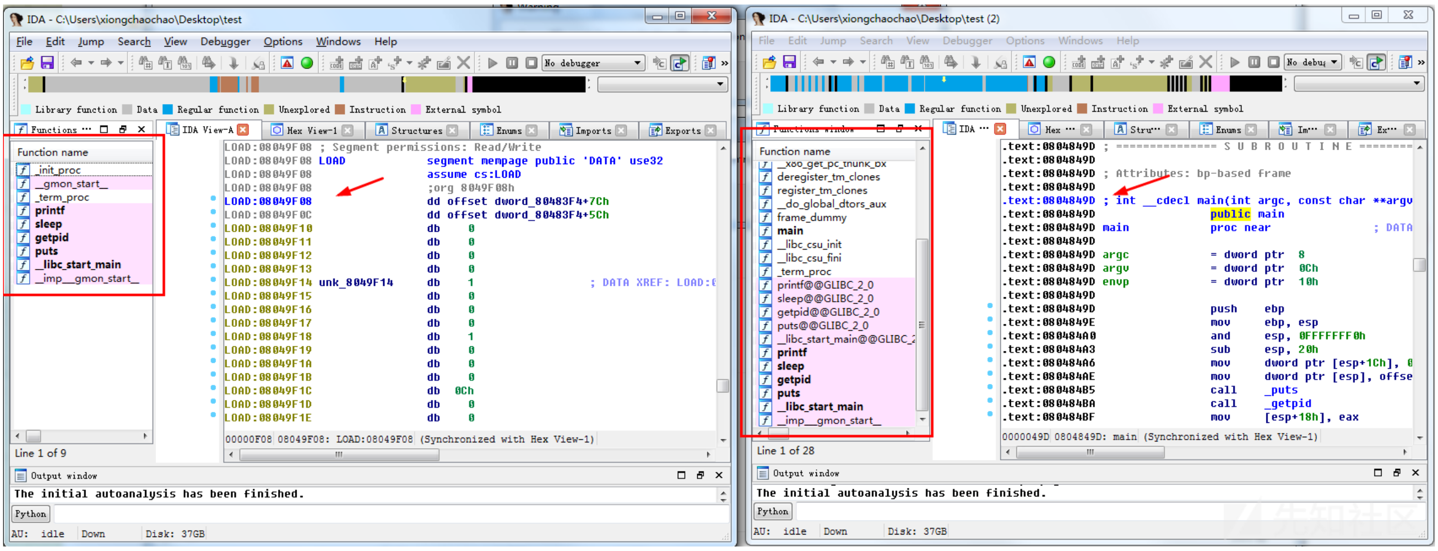
```
#include <stdio.h>
#include <elf.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>



int main(int argc, char** argv)
{
    int fd;
    Elf32_Ehdr ehdr;
    Elf32_Shdr shdr;
    char * section_name_table;
    int i;
    unsigned int base, length;
    char *content;
    unsigned short nsize;


    //■■■■
    if(argc != 3)
    {
        printf("Encrypt section of elf file\n\nUsage:\n\t%s <elf_file> <section_name>\n", *argv);
        goto _error;
    }

    if((fd = open(argv[1], O_RDWR, 0777)) == -1)
    {
        perror("open");
        goto _error;
}

if(read(fd, &ehdr, sizeof(Elf32_Ehdr)) != sizeof(Elf32_Ehdr))
{
    perror("read elf header");
    goto _error;
}

//■■■■■■■■
printf("[+] Begining find section %s\n", argv[2]);
lseek(fd, ehdr.e_shoff+sizeof(Elf32_Shdr)*ehdr.e_shstrndx, SEEK_SET);
if(read(fd, &shdr, sizeof(Elf32_Shdr)) != sizeof(Elf32_Shdr))
{
    perror("read elf section header which contain string table");
    goto _error;
}

if((section_name_table = (char*) malloc(shdr.sh_size)) == NULL)
{
    perror("malloc for SHT_STRTAB");
    goto _error;
}

lseek(fd, shdr.sh_offset, SEEK_SET);
if(read(fd, section_name_table, shdr.sh_size) != shdr.sh_size)
{
    perror("read string table");
    goto _error;
}
lseek(fd, ehdr.e_shoff, SEEK_SET);
//■■■■■■■■■■■■■■■
for(i=0; i<ehdr.e_shnum; i++)
{
    if(read(fd, &shdr, sizeof(Elf32_Shdr)) != sizeof(Elf32_Shdr))
    {
        perror("read section");
        goto _error;
    }
    if(strcmp(section_name_table+shdr.sh_name, argv[2]) == 0)
    {
        base = shdr.sh_offset;
        length = shdr.sh_size;
        printf("[+] Find section %s\n", argv[2]);
        printf("[+] %s section offset is %X\n", argv[2], base);
        printf("[+] %s section size is %d\n", argv[2], length);
        break;
    }
}

//■■■■■■
lseek(fd, base, SEEK_SET);
content = (char *)malloc(length);
if(content == NULL)
{
    perror("malloc space for section");
    goto _error;
}
if(read(fd, content, length) != length)
{
    perror("read section in encrpt");
    goto _error;
}
//■■■■■■■■■■■■■■■
//■■■■■■■■■■■■■■■■■■■■■■■■
nsize = length/4096 + (length%4096 == 0 ? 0 : 1);
ehdr.e_entry = (length << 16) + nsize;
ehdr.e_shoff = base;
printf("[+] %s section use %d memory page!\n", argv[2], nsize);
```

```
    //■■■■
    for(i=0; i<length; i++)
    {
        content[i] = ~content[i];
    }

    lseek(fd, 0, SEEK_SET);
    if(write(fd, &ehdr, sizeof(Elf32_Ehdr)) != sizeof(Elf32_Ehdr))
    {
        perror("write ELF header to file");
        goto _error;
    }
    lseek(fd, base, SEEK_SET);
    if(write(fd, content, length) != length)
    {
        perror("write encrypted section to file");
        goto _error;
    }

    printf("[+] Encrypt section %s completed!\n", argv[2]);
_error:
    free(section_name_table);
    free(content);
    close(fd);
    return 0;
}
```

利用上面代码加密后32位ELF文件用IDA打开就会出现以下错误，不能进入程序正确入口并且不能从节头、函数符号也收到影响



Illegal program entry point (1D20001)


IDA - C:\Users\xiongchaochao\Desktop\test
File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Data Regular function Unexplored Instruction External symbol
Functions ···
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
Output window
Python
IDA - C:\Users\xiongchaochao\Desktop\test (2)
Functions window
IDA ···
Hex ···
Stru···
Im···
Ex···

```
Function name
_init_proc
__gmon_start__
_term_proc
printf
sleep
getpid
puts
__libc_start_main
__imp___gmon_start__

Line 1 of 9
```

```
LOAD:08049F08 ; Segment permissions: Read/Write
LOAD:08049F08 LOAD            segment mempage public 'DATA' use32
LOAD:08049F08                 assume cs:LOAD
LOAD:08049F08                 ;org 8049F08h
LOAD:08049F08                 dd offset dword_80483F4+7Ch
LOAD:08049F0C                 dd offset dword_80483F4+5Ch
LOAD:08049F10                 db    0
LOAD:08049F11                 db    0
LOAD:08049F12                 db    0
LOAD:08049F13                 db    0
LOAD:08049F14 unk_8049F14     db    1                   ; DATA XREF: LOAD:0
LOAD:08049F15                 db    0
LOAD:08049F16                 db    0
LOAD:08049F17                 db    0
LOAD:08049F18                 db    1
LOAD:08049F19                 db    0
LOAD:08049F1A                 db    0
LOAD:08049F1B                 db    0
LOAD:08049F1C                 db  0Ch
LOAD:08049F1D                 db    0
LOAD:08049F1E                 db    0
00000F08 08049F08: LOAD:08049F08 (Synchronized with Hex View-1)
```

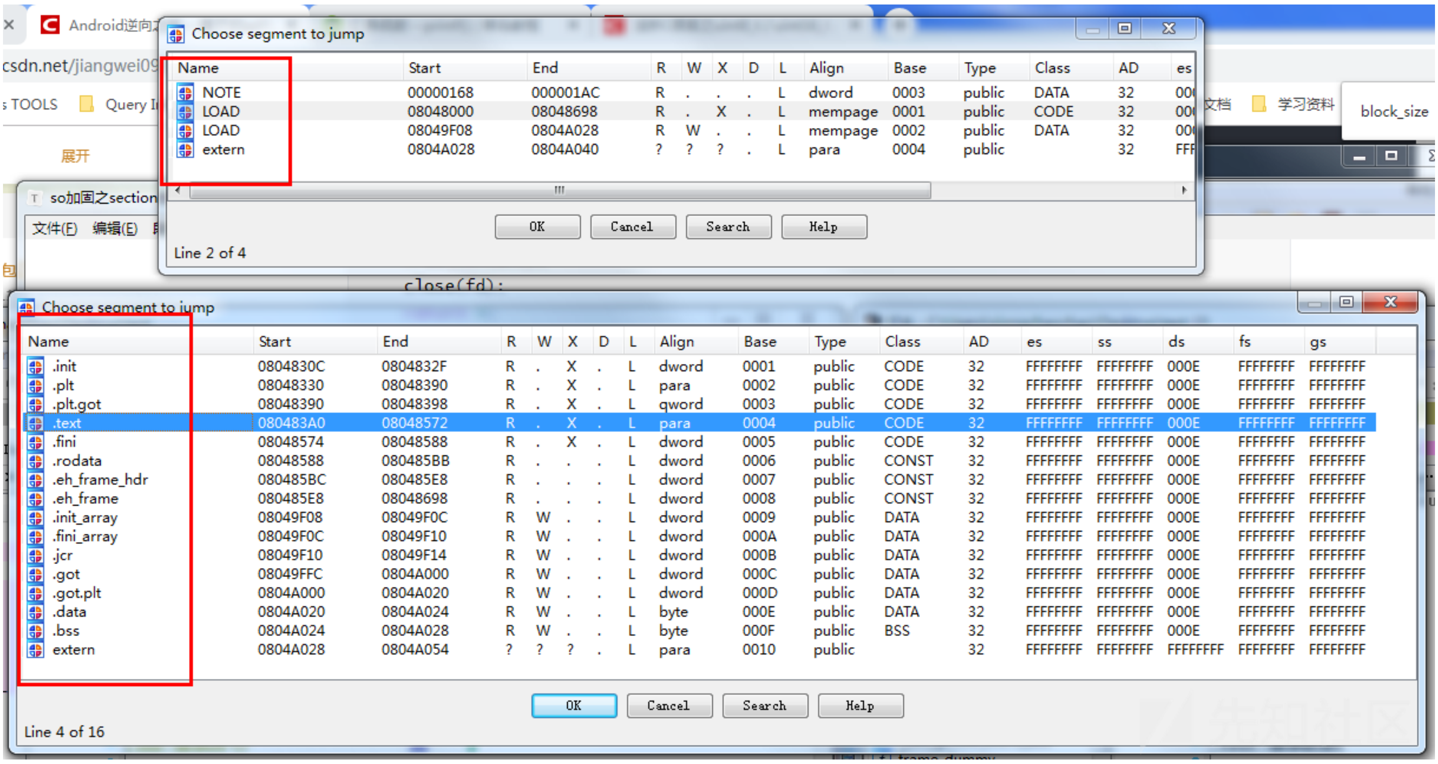
```
The initial autoanalysis has been finished.
AU:  idle   Down      Disk: 37GB
```

```
Function name
__x86_get_pc_thunk_bx
deregister_tm_clones
register_tm_clones
__do_global_dtors_aux
frame_dummy
main
__libc_csu_init
__libc_csu_fini
_term_proc
printf@@GLIBC_2_0
sleep@@GLIBC_2_0
getpid@@GLIBC_2_0
puts@@GLIBC_2_0
__libc_start_main@@GLIBC_2
printf
sleep
getpid
puts
__libc_start_main
__imp___gmon_start__

Line 1 of 28
```

```
.text:0804849D ; =============== S U B R O U T I N E =======
.text:0804849D
.text:0804849D ; Attributes: bp-based frame
.text:0804849D
.text:0804849D ; int __cdecl main(int argc, const char **argv
.text:0804849D                 public main
.text:0804849D main            proc near               ; DATA
.text:0804849D
.text:0804849D argc            = dword ptr  8
.text:0804849D argv            = dword ptr  0Ch
.text:0804849D envp            = dword ptr  10h
.text:0804849D
.text:0804849D                 push    ebp
.text:0804849E                 mov     ebp, esp
.text:080484A0                 and     esp, 0FFFFFFF0h
.text:080484A3                 sub     esp, 20h
.text:080484A6                 mov     dword ptr [esp+1Ch], 0
.text:080484AE                 mov     dword ptr [esp], offse
.text:080484B5                 call    _puts
.text:080484BA                 call    _getpid
.text:080484BF                 mov     [esp+18h], eax
0000049D 0804849D: main (Synchronized with Hex View-1)
```

```
The initial autoanalysis has been finished.
AU:  idle   Down      Disk: 37GB
```

节头完全识别不出来只能用段表来显示



Choose segment to jump

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	es
NOTE	00000168	000001AC	R	.	.	.	L	dword	0003	public	DATA	32	00
LOAD	08048000	08048698	R	.	X	.	L	mempage	0001	public	CODE	32	00
LOAD	08049F08	0804A028	R	W	.	.	L	mempage	0002	public	DATA	32	00
extern	0804A028	0804A040	?	?	?	.	L	para	0004	public		32	FFF

Line 2 of 4

Choose segment to jump

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	es	ss	ds	fs	gs
.init	0804830C	0804832F	R	.	X	.	L	dword	0001	public	CODE	32	FFFFFFFF	FFFFFFFF	000E	FFFFFFFF	FFFFFFFF
.plt	08048330	08048390	R	.	X	.	L	para	0002	public	CODE	32	FFFFFFFF	FFFFFFFF	000E	FFFFFFFF	FFFFFFFF
.plt.got	08048390	08048398	R	.	X	.	L	qword	0003	public	CODE	32	FFFFFFFF	FFFFFFFF	000E	FFFFFFFF	FFFFFFFF
.text	080483A0	08048572	R	.	X	.	L	para	0004	public	CODE	32	FFFFFFFF	FFFFFFFF	000E	FFFFFFFF	FFFFFFFF
.fini	08048574	08048588	R	.	X	.	L	dword	0005	public	CODE	32	FFFFFFFF	FFFFFFFF	000E	FFFFFFFF	FFFFFFFF
.rodata	08048588	080485BB	R	.	.	.	L	dword	0006	public	CONST	32	FFFFFFFF	FFFFFFFF	000E	FFFFFFFF	FFFFFFFF
.eh_frame_hdr	080485BC	080485E8	R	.	.	.	L	dword	0007	public	CONST	32	FFFFFFFF	FFFFFFFF	000E	FFFFFFFF	FFFFFFFF
.eh_frame	080485E8	08048698	R	.	.	.	L	dword	0008	public	CONST	32	FFFFFFFF	FFFFFFFF	000E	FFFFFFFF	FFFFFFFF
.init_array	08049F08	08049F0C	R	W	.	.	L	dword	0009	public	DATA	32	FFFFFFFF	FFFFFFFF	000E	FFFFFFFF	FFFFFFFF
.fini_array	08049F0C	08049F10	R	W	.	.	L	dword	000A	public	DATA	32	FFFFFFFF	FFFFFFFF	000E	FFFFFFFF	FFFFFFFF
.jcr	08049F10	08049F14	R	W	.	.	L	dword	000B	public	DATA	32	FFFFFFFF	FFFFFFFF	000E	FFFFFFFF	FFFFFFFF
.got	08049FFC	0804A000	R	W	.	.	L	dword	000C	public	DATA	32	FFFFFFFF	FFFFFFFF	000E	FFFFFFFF	FFFFFFFF
.got.plt	0804A000	0804A020	R	W	.	.	L	dword	000D	public	DATA	32	FFFFFFFF	FFFFFFFF	000E	FFFFFFFF	FFFFFFFF
.data	0804A020	0804A024	R	W	.	.	L	byte	000E	public	DATA	32	FFFFFFFF	FFFFFFFF	000E	FFFFFFFF	FFFFFFFF
.bss	0804A024	0804A028	R	W	.	.	L	byte	000F	public	BSS	32	FFFFFFFF	FFFFFFFF	000E	FFFFFFFF	FFFFFFFF
extern	0804A028	0804A054	?	?	?	.	L	para	0010	public		32	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF

Line 4 of 16

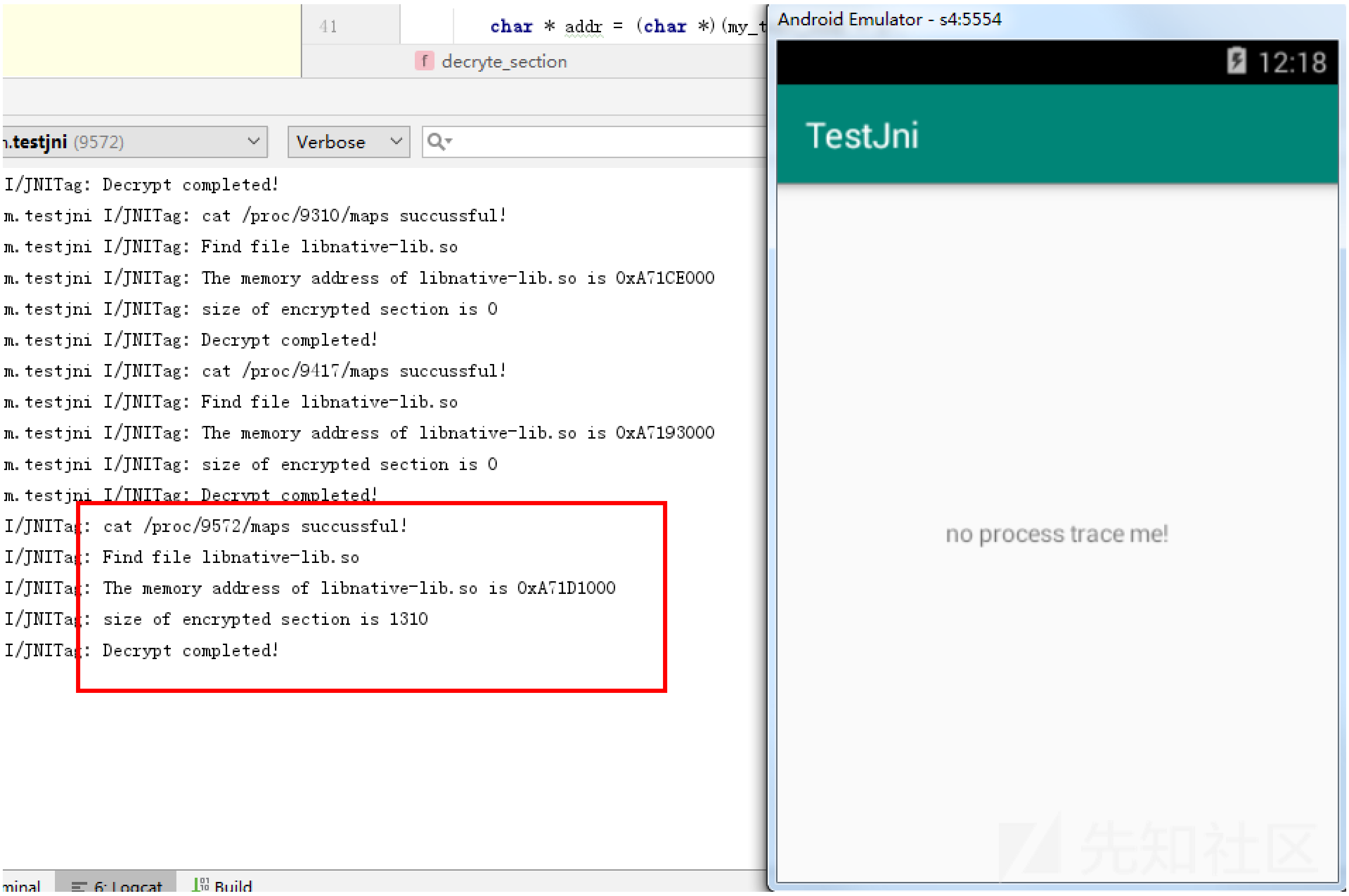
Native解密

解密原理：我们从加密后so文件头的入口点右移16的值中获取加密的自定义节的长度，从文件头中的节头表偏移值中获取加密的节的内存偏移。然后我们用mprotect把这个

```
#include <jni.h>
#include <string>
#include <asm/fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sstream>
#include <fcntl.h>
#include <android/log.h>
#include <elf.h>
#include <sys/mman.h>

#define PAGE_SIZE 4096

jstring getString(JNIEnv*) __attribute__((section (".mytext")));
//■■■■■■■■■■init_array■■■
void decryte_section() __attribute__((constructor));
unsigned long getLibAddr();

void decryte_section() {
   unsigned long base;
   Elf32_Ehdr *ehdr;
   Elf32_Shdr *shdr;
   unsigned long my_text_addr;
   unsigned int nblock;
   unsigned int nsize;
   unsigned int i;


   base = getLibAddr();
   ehdr = (Elf32_Ehdr *)base;
   //■■■■■■■■
   my_text_addr = base + ehdr->e_shoff;
   nblock = ehdr->e_entry >> 16;
   nsize = (nblock / PAGE_SIZE) + (nblock%PAGE_SIZE == 0 ? 0 : 1);
   __android_log_print(ANDROID_LOG_INFO, "JNITag", "size of encrypted section is %d", nblock);
   if (mprotect((void *)(my_text_addr / PAGE_SIZE * PAGE_SIZE), nsize*PAGE_SIZE, PROT_READ | PROT_EXEC | PROT_WRITE) == -1){
       __android_log_print(ANDROID_LOG_ERROR, "JNITag", "Memory privilege change failed before encrypt");
   }
   //■■
```

```
    for(i=0; i<nblock; i++){
        char * addr = (char *)(my_text_addr + i);
        * addr = ~(*addr);
    }
    //■■■■■■■
    if (mprotect((void *)(my_text_addr / PAGE_SIZE * PAGE_SIZE), nsize*PAGE_SIZE, PROT_READ | PROT_EXEC) == -1){
        __android_log_print(ANDROID_LOG_ERROR, "JNITag", "Memory privilege change failed after encrypt");
    }
    __android_log_print(ANDROID_LOG_INFO, "JNITag", "Decrypt completed!");
}

/** ■■■■■■■■■■■■■■■■■■■■■■*/
unsigned long getLibAddr(){

    int pid;
    char buffer[4096];
    FILE *fd;
    char *tmp;

    unsigned long ret = 0;
    char so_name[] = "libnative-lib.so";

    pid = getpid();
    sprintf(buffer, "/proc/%d/maps", pid);
    if((fd = fopen(buffer, "r")) == NULL){
        __android_log_print(ANDROID_LOG_DEBUG, "JNITag", "open /proc/%d/maps failed!", pid);
        goto _error;
    }
    __android_log_print(ANDROID_LOG_INFO, "JNITag", "cat /proc/%d/maps succussful!", pid);
    while(fgets(buffer, sizeof(buffer), fd)){
        if(strstr(buffer, so_name)){
            tmp = strtok(buffer, "-");
            ret = strtoul(tmp, 0, 16);
            __android_log_print(ANDROID_LOG_INFO, "JNITag", "Find file %s", so_name);
            __android_log_print(ANDROID_LOG_INFO, "JNITag", "The memory address of %s is 0x%X", so_name, ret);
            break;
        }
    }

_error:
    fclose(fd);
    return ret;
}

//■■■java■■■■Native■■
extern "C"
JNIEXPORT jstring JNICALL
Java_com_testjni_MainActivity_isTraceMe(JNIEnv *env, jobject instance){
    return getString(env);
}

//■■■■■■■■■■■■■■■■■■■■■
jstring getString(JNIEnv* env){
    return env->NewStringUTF("Text from JNI");
};
```

从下面可以运行结果显示，实现了自定义节的动态解密过程



```
char * addr = (char *)(my_t
```

```
I/JNITag: Decrypt completed!
m.testjni I/JNITag: cat /proc/9310/maps succussful!
m.testjni I/JNITag: Find file libnative-lib.so
m.testjni I/JNITag: The memory address of libnative-lib.so is 0xA71CE000
m.testjni I/JNITag: size of encrypted section is 0
m.testjni I/JNITag: Decrypt completed!
m.testjni I/JNITag: cat /proc/9417/maps succussful!
m.testjni I/JNITag: Find file libnative-lib.so
m.testjni I/JNITag: The memory address of libnative-lib.so is 0xA7193000
m.testjni I/JNITag: size of encrypted section is 0
m.testjni I/JNITag: Decrypt completed!
I/JNITag: cat /proc/9572/maps succussful!
I/JNITag: Find file libnative-lib.so
I/JNITag: The memory address of libnative-lib.so is 0xA71D1000
I/JNITag: size of encrypted section is 1310
I/JNITag: Decrypt completed!
```

小结

本篇文章主要写了如何对section的加密、以及在`.init_array`节中进行动态解密的详细过程。

想要绕过也是可以的，通过动态调试在解密的.init_array节处下断点，然后dump出解密后的so文件进行反编译即可

参考

[0] Android so库加固加壳方案

[1] Android逆向之旅---基于对so中的section加密技术实现so加固

[3] [原创]简单粗暴的so加解密实现

点击收藏 | 0 关注 | 1

上一篇：如何利用机器学习创建恶意软件检测系统 下一篇：Laravel 5.8 SQL 注...

1. 2 条回复

shaomi 2019-06-06 17:13:20

“在.initarray节处下断点”这个只是里面有反调试代码才需要下断点....没反调试的普通段处理话，根本不用断点的，只要attach了，啥时候dump都能dump出so的，只要是

1 回复Ta

yong夜 2019-06-07 11:21:40

@shaomi 仔细一想确实是这样的，谢谢大佬提醒 :)

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板