

 NIKE.COM	 NIKE SOCK DART SE 女子运动鞋	 NIKE AIR FORCE 1 ULTRA FORCE MID 女子运动鞋	前往 NIKE.COM
会员专享 免费送货服务			

行者无疆-超越的博客

http://blog.sina.com.cn/u/3672674513 [订阅] [手机订阅]

首页 博文目录 图片 关于我

个人资料

推荐: 拿到150封offer的简历 虽然相亲失败却网恋成功



博客目录



行者无疆-超越

发纸条

加关注

篷房报价



博客等级: 
博客积分: **53**
博客访问: **11,595**
关注人气: **1**
获赠金笔: **0**
赠出金笔: **0**
荣誉徽章:

相关博文

- 实拍: 古村留守儿童的春节大山的孩子
- 街拍: 穿丝袜露大腿最拉风曹作兰
- 第一章: 万夫莫当巨厥
- 地暖材料大盘点克罗顿维尔三招选用户314067242

Android利用ptrace实现Hook API (2014-04-23 18:55:34)

标签: android hookapi ptrace it 分类: Android安全

Hook API的技术由来已久，在操作系统未能提供所需功能的情况下，利用Hook API的手段来实现某种必需的功能也是一种不得已的办法。

笔者了解Hook API技术最早是在十几年前，当时是在Windows平台下开发电子词典的光标取词功能。用Hook API的技术把系统的字符串输出函数替换成了电子词典中的函数，从而能得到屏幕上任何位置的字符串。无论是16位的Windows95还是32的Windows NT都有办法向整个系统或特定的目标进程中“注入”DLL动态库，并替换其中的函数。

Linux平台中完成Hook API的方法和Windows上不一样。Linux由于安全性更高，所以一般的方法难以达到目标，通常是采用ptrace函数来实现Hook API的目的。但是调用ptrace函数需要root权限，所以开发的软件作用有限。这也是为什么Hook API技术在Linux上并不流行。

Android上最早使用Hook API技术的软件是“xx安全大师”，因为使用了这项“秘密武器”，所以完成了很多看起来不可思议的功能。当然现在在国内市场上差不多所有的安全类软件都实现了类似的功能。和Linux下一样，使用这些功能的前提是获得root授权。

这些安全软件因为要截获系统的binder功能，所以多半是替换掉libc库的ioctl函数来达到监控binder调用的目的。下面我们也来学习如何用自己开发的动态库中的ioctl函数替换目标进程中的ioctl函数。

原理看起来并不复杂，但是实现起来却不是那么的简单，向目标进程中注入代码的步骤是：

- 1) 用ptrace函数attach上目标进程；
- 2) 让目标进程的执行流程跳转到mmap函数来分配一小段内存空间；
- 3) 把一段机器码拷贝到目标进程中刚分配的内存中去；
- 4) 最后让目标进程的执行流程跳转到注入的代码执行。

下面将详细的介绍具体的过程。

1. attach目标进程

用ptrace函数attach上目标进程的代码如下：

```
ptrace( PTRACE_ATTACH, pid, NULL, 0 );
```

在继续操作前，需要先把目标进程的寄存器先保存起来，这样完成注入后，恢复目标进程的寄存器，目标进程就能不受影响继续执行了。

```
struct pt_regs old_regs;
ptrace( PTRACE_GETREGS, pid, NULL, &old_regs );
```

2. 获取目标进程中函数地址

为了在目标进程中调用mmap函数，需要得到mmap函数在目标进程中的地址。由于函数地址相对于动态库的装载地址是固定不变的，所以本进程mmap函数的地址加上目标进程和本进程的动态库的装载地址的差值就等于目标进程的中mmap函数的地址。用公式表达如下：

目标进程函数地址 = 本进程函数地址 + （本进程libc库地址 - 目标进程lib库地址）

这样其实只要得到动态库的装载地址就能算出目标进程的mmap的地址。一种得到动态库装载地址的方法是分析Linux进程的/proc/pid/maps文件，这个文件包含了进程中所有mmap映射的地址。下面我们写一个获取动态库地址的函数，代码如下：

```
unsigned long get_lib_address( pid_t pid, const char* library_name )
{
    char filename[256];
    if ( pid < 0 ) { // 得到当前进程模块地址时传入的pid参数为-1
        sprintf( filename, sizeof(filename), "/proc/self/maps" );
    }
```

- 先生，别慌！冬季这样戴手表才是深圳钟表展
- 冬天睡眠才是第一大补要想睡眠好，英国斯林百兰床垫
- 埋线双眼皮能保持多久？整形咨询师-田文
- 今晚英联杯曼联vs赫尔城 用户333279115
- 青衫老祖：此“一箭三星”与彼“九州虫
- 蓝唐度假酒店周边旅游度假攻略 用户370936980
- 2016，中国“国运”来了挡不住？大浪爱莎
- 【蓝色大海的传说】《蓝海传说》韩剧疯子

推荐：拿到150封offer的简历 虽然相亲失败却网恋成功

[新浪首页](#) [登录](#) [注册](#)



- 推荐博文
- 新零售拓展阿里发展空间&nb
- 20170223【早评】短线需
- 董明珠真的对做新能源汽车这么执
- 台湾科技挣扎，人祸大于天灾？
- 木子析：绩效考核切忌“一锅粥”
- 收入份额=市场份额，虎嗅想干什
- 传奇的谢幕，谈岩田聪和他的任天堂
- 家常主食轻松做之一——培根香葱花
- 共享单车的押金去哪，没必要告诉
- 盘点2015最惊艳流行的婚礼蛋

[查看更多>>](#)

```
    } else {
        snprintf( filename, sizeof(filename), "/proc/%d/maps", pid );
    }
    FILE *fp = fopen( filename, "r" );
    if ( fp != NULL ) {
        char line[1024];
        while ( fgets( line, sizeof(line), fp ) ) {
            // 在所有的映射行中寻找目标动态库所在的行
            if ( strstr( line, library_name ) ) {
                char *p = strtok( line, "-" );
                unsigned long addr = strtoul(p, NULL, 16 );
                fclose( fp );
                return addr;
            }
        }
        fclose( fp );
    }
    return 0;
}
```

有了这个函数，我们就能算出mmap函数在目标进程中所在的地址了。用前面介绍的方法计算函数地址的代码如下所示。

```
unsigned long remote_address = get_lib_address( pid, "/system/lib/libc.so" );
mmap_addr= (unsigned long)mmap + remote_address - local_address ;
```

3. 调用目标进程中的函数

调用函数是需要传递参数的，arm中前4个参数可以使用寄存器传递，后面的参数需要通过栈传递，mmap有6个参数，所以我们要向目标进程的栈中写入参数。

```
long params [] = { // 调用mmap函数的参数
    0,
    MAP_SIZE,
    PROT_READ | PROT_WRITE | PROT_EXEC,
    MAP_ANONYMOUS | MAP_PRIVATE,
    0,
    0
};
// 前面四个参数用寄存器传递
for ( i = 0; i < 4; i ++ ) {
    regs->uregs[i] = params[i];
}
regs->ARM_sp -= 2* sizeof(long) ;

// 后面两个参数放到栈里
unsigned long addr = regs->ARM_sp;
ptrace( PTRACE_POKETEXT, pid, addr, params[5]);
ptrace( PTRACE_POKETEXT, pid, addr+sizeof(long), params[6]);
```

让目标进程执行函数的方法就是把目标进程的pc寄存器设为函数的入口地址，然后让目标进程恢复运行就可以了。但是函数执行完还需要让本进程继续控制，为了达到这个目的，这里使用的一点技巧：调用mmap时把返回地址设为0，这样目标进程执行完mmap返回时会出现地址错误，这样目标进程将被挂起，控制权会回到调试进程手中，现在的调试进程就是我们的应用程序。

```
regs->ARM_pc = mmap_addr; // 设置pc寄存器为mmap函数的入口
regs->ARM_lr = 0; // 把返回地址置为0
ptrace( PTRACE_SETREGS, pid, NULL, regs ); // 设置目标进程的寄存器
ptrace( PTRACE_CONT, pid, NULL, 0 ); // 让目标进程继续运行
waitpid( pid, NULL, WUNTRACED ); // 等待目标进程结束
```

调用结束后，需要读取目标进程的寄存器，其中寄存器r0保存着返回值，也就是mmap分配的地址：

```
ptrace( PTRACE_GETREGS, pid, NULL, regs);
unsigned long remote_mmap_base = (unsigned long)regs.ARM_r0;
```

4. 注入代码并运行

mmap运行完成后，我们已经在目标进程中拥有了一块内存，现在只需要把准备好的代码用ptrace写进去就可以了。因此关键的问题是如何准备这段代码。

需要注入的代码是用来装载我们自己的动态库的，并且装载完毕后还要能调用其中的函数hook_api来完成替换系统ioctl函数的工作，最后还要将目标进程恢复到初始状态运行，就好像什么也没发生过。

我们需要用汇编来写这段代码：

```
.global _dlopen_addr_s
.global _dlopen_param1_s
.global _dlopen_param2_s

.global _dlsym_addr_s
.global _dlsym_param2_s
```

```

.global _saved_cpsr_s
.global _saved_regs_s

.data
_code_start_s:

@执行dlopen函数
ldr r1, _dlopen_param2_s
ldr r0, _dlopen_param1_s
ldr r3, _dlopen_addr_s
blx r3

@调用dlclose函数得到函数hook_api的地址
ldr r1, _dlsym_param2_s
ldr r3, _dlsym_addr_s
blx r3

@调用hook_api函数
blx r0

```

@恢复原始的cpsr和寄存器值

推荐: 拿到150封offer的简历 虽然相亲失败却网恋成功 [×](#)

[新浪首页](#) [登录](#) [注册](#)

```

msr cpsr_c1, r1
ldr sp, _saved_r0_pc_s
ldmfd sp, {r0-pc}

```

```

_dlopen_addr_s:
.word 0xFFFFFFFF
_dlopen_param1_s:
.word 0xFFFFFFFF
_dlopen_param2_s:
.word 0x2
_dlsym_addr_s:
.word 0xFFFFFFFF
_dlsym_param2_s:
.word 0xFFFFFFFF
_saved_cpsr_s:
.word 0xFFFFFFFF
_saved_regs_s:
.word 0xFFFFFFFF
_code_end_s:
.space 0x400, 0
.end

```

这段汇编代码放在了.data段中，所以定义的并不是函数，只是代码片段，这样的好处是把编译后的二进制代码注入到目标进程就可以运行。

同时代码中还定义了一些变量，包括函数dlopen和dlsym的地址也是用变量来表示的，这是因为我们是在自己的应用中完成这段汇编代码编译的，如果注入到目标进程中，函数的地址并不相同，所以把函数地址用变量表示出来，在注入前需要计算出dlopen和dlsym在目标进程中的地址，填在这里。

变量_dlopen_param1_s用来定义dlopen函数的第一个参数，也就是动态库的路径，所以库的路径字符串也需要拷贝到目标进程中，因此代码的最后通过语句“space 0x400,0”开辟了一段空间来存储路径字符串等参数。

变量_dsym_param2_s用来定义dlsym函数的第二个参数：需要调用的函数名，它也需要拷贝到目标进程中。

变量_saved_cpsr_s和_saved_regs_s用来保存目标进程原始的cpsr和寄存器值，这样当dlopen函数返回后，通过恢复cpsr和寄存器就能让目标进程恢复运行了。

下面的代码演示了如何初始化上面这些变量：

```

//用前面介绍过的方法得到目标进程中dlopen的地址并填入变量_dlopen_addr_s中
unsigned long local_handle = get_lib_address( -1, "/system/lib/linker" );
unsigned long remote_handle = get_lib_address( pid, "/system/lib/linker " );
_dlopen_addr_s = (unsigned long)dlopen + remote_handle - local_handle ;
_dlsym_addr_s = (unsigned long)dlsym + remote_handle - local_handle ;

// 变量remote_code_ptr存储的是目标进程中注入代码的起始地址，
// 但是我们要留出一段空间作为调用dlopen的栈，所以并没有把起始地址定为mmap地址的开头，
// 而是加上了一个堆栈的长度
unsigned long remote_code_ptr = remote_mmap_base + STACK_SIZE;

// 变量local_code_ptr指向汇编中的代码开始地址
unsigned long local_code_ptr = (unsigned long)&_code_start_s;

// 计算代码的长度
int lcode_length = (unsigned long)&_code_end_s - (unsigned long)&_code_start_s;

// 变量dlopen_param1_ptr指向汇编中保留的空间，用来存储动态库的路径

```



```

unsigned long dlopen_param1_ptr = local_code_ptr + code_length + 0x40;

// 变量dlsym_param2_ptr指向汇编中保留的空间, 用来存储函数名字符串
unsigned long dlsym_param2_ptr = dlopen_param1_ptr + 0x100;

// 变量saved_regs_ptr指向汇编中保留的空间, 用来存储原始的寄存器
unsigned long saved_regs_ptr = dlsym_param2_ptr + 0x100;

// 拷贝动态库的路径字符串
strcpy((char*)dlopen_param1_ptr, "/system/lib/libhook.so");

// 计算路径字符串在目标进程中的地址
_dlopen_param1_s = dlopen_param1_ptr + remote_code_ptr - local_code_ptr);

// 拷贝函数名字符串
strcpy((char*)dlsym_param2_ptr, "hook_api");

// 计算函数名字符串在目标进程中的地址
_dlsym_param2_s = dlsym_param2_ptr + remote_code_ptr - local_code_ptr);

// 把目标进程原始的寄存器值r0 ~ r15拷贝到变量saved_regs_ptr中
memcpy((void*)saved_regs_ptr, &(old_regs.ARM_r0), 16 * sizeof(long));

// 计算保存寄存器的变量在目标进程中的地址
_saved_regs_s = saved_regs_ptr + remote_code_ptr - local_code_ptr);

```

推荐: 拿到150封offer的简历 虽然相亲失败却网恋成功 [×](#)

[新浪首页](#) [登录](#) [注册](#)

```

// 把目标进程原始的寄存器值r0 ~ r15拷贝到变量saved_regs_ptr中
memcpy((void*)saved_regs_ptr, &(old_regs.ARM_r0), 16 * sizeof(long));

// 计算保存寄存器的变量在目标进程中的地址
_saved_regs_s = saved_regs_ptr + remote_code_ptr - local_code_ptr);
初始化这些变量用了很多编程技巧, 不太容易理解, 所以笔者在这里每行都做了注释。
最后, 把准备好的代码“拷贝”到目标进程中并运行:
// ptrace_writedata把一段内存“拷贝”到目标进程的函数
ptrace_writedata(pid, remote_code_ptr, (char*)local_code_ptr, MAP_SIZE-STACK_SIZE);
memcpy(&regs, &old_regs, sizeof(regs)); // 准备寄存器
regs.ARM_sp = (long)remote_code_ptr; // 初始化堆栈
regs.ARM_pc = (long)remote_code_ptr; // 把PC寄存器设为代码的入口地址
ptrace(PTRACE_SETREGS, pid, NULL, &regs); // 设置目标进程的寄存器
ptrace_detach(pid); // 退出控制, 这样目标进程就可以恢复运行了

```

上面的代码中用ptrace_writedata函数来拷贝一块内存到目标进程中。这个函数只是封装了ptrace函数来一次拷贝更多的数据, 这里就不多介绍了。

这样我们终于在目标进程中装载进了我们开发的动态库, 这个库里有个名为new_ioctl的函数, 它就是我们准备用来替换系统ioctl的函数, 下面我们将介绍替换过程。

5. 替换系统的ioctl函数

在前面linker的介绍中读者应该已经了解了, 每个动态库都有自己的全局偏移表GOT。而我们希望完成的是将binder函数所在的库libbinder.so中的ioctl函数替换掉, 打算不打算替换所有动态库中的ioctl调用, 所以我们只要找到libbinder.so的全局偏移表就达到目标了。代码如下:

```

bool hook_api() {
    // 使用打开动态库的方式得到动态库的soinfo结构
    soinfo* si = (soinfo*):dlopen("/system/bin/libbinder.so", RTLD_NOW);
    if(si == NULL || si->strtab == NULL || si->plt_rel == NULL) {
        return false;
    }
    for (uint32_t i = 0; i < si->plt_rel_count; i++) {
        // 查找重定位表中ioctl所在的项
        if(strcmp(si->symtab[ELF32_R_SYM(si->plt_rel[i].r_info)].st_name
            + si->strtab, "ioctl") == 0) {
            // 计算对应的GOT表项的地址
            uint32_t* got = (uint32_t*)(si->base + si->plt_rel[i].r_offset);
            if(*got != new_ioctl) {
                // 把GOT表项的地址属性改为可写
                uint32_t pagesize = sysconf(_SC_PAGE_SIZE);
                void* start = (void*)((uint32_t)got)/pagesize*pagesize;
                if (mprotect(start, pagesize * 2, PROT_READ|PROT_WRITE) == -1) {
                    return false;
                }
                *(got) = new_ioctl; // 填入新地址
                mprotect(start, pagesize * 2, PROT_READ|PROT_WRITE);
            }
            return true;
        }
    }
}

```



查找ioctl在GOT表中的地址是通过查找动态库的函数重定位表来完成的。前面介绍linker模块时对重定位表已经解释的很详细了，这里的代码就不用多解释了。

为了节省篇幅，本节中的代码很多都去掉了错误判断和处理语句，读者如果要借鉴这些代码，要注意把这部分代码补全了。这里介绍了注入部分，其实后面的binder替换函数的编写，各种系统调用的处理也非常麻烦，需要对Android的Binder机制和framework有深入的了解才能完成。

3

喜欢

0

赠金笔

分享：

推荐：拿到150封offer的简历 虽然相亲失败却网恋成功

新浪首页 登录 注册

前一篇：[Bionic中的ptrace函数](#)
后一篇：[Android ART模式简介](#)

评论

重要提示：警惕虚假中奖信息

[发评论]



做第一个评论者吧！ 抢沙发>>

发评论

更多>>

登录名： 密码： [找回密码](#) [注册](#) ☒ 记住登录状态

☒ 分享到微博 ☐ 评论并转载此博文

按住左边滑块，拖动完成上方拼图

发评论

以上网友发言只代表其个人观点，不代表新浪网的观点或立场。

< 前一篇


[Bionic中的ptrace函数](#)

后一篇 >

[Android ART模式简介](#)

推荐：拿到150封offer的简历 虽然相亲失败却网恋成功 ×

[新浪首页](#) [登录](#) [注册](#)



篷房报价

