XNU 审计笔记

Brightiup

2018-03-28

目录

1 系统调用 **1**

1.1 系统调用种类以及入口 1

2 文件系统 **2**

3 网络系统 **2**

4 内存管理 **2**

5 进程管理 **2**

6 Mach 相关 **2**

7 BSD 相关 **2**

8 IOKit **3**

8.1 IOUserClient 对象生命周期 3

9 缓解策略 **6**

1 系统调用

1.1 系统调用种类以及入口

xnu 总共有四种系统调用，如表 1。

系统调用种类	函数入口
unix	hndl_unix_scall64
mach	hndl_mach_scall64
machdep	hndl_mdep_scall64
diagnostics	hndl_diag_scall64

表 1: xnu 系统调用种类

通过 *syscall* 指令可以直接进入这几个函数。*syscall* 指令的处理入口为 *hndl_syscalll*，其实现如下：

```
// xnu/osfmk/x86_64/idt64.s
/*
 * 64bit Tasks
 * System call entries via syscall only:
 *
 *  r15  x86_saved_state64_t
 *  rsp  kernel stack
 *
 *  both rsp and r15 are 16—byte aligned
 *  interrupts disabled
 *  direction flag cleared
 */

Entry(hndl_syscall)
    TIME_TRAP_UENTRY

    movq    %gs:CPU_ACTIVE_THREAD,%rcx  /* get current thread     */
    movl    $—1, TH_IOTIER_OVERRIDE(%rcx)   /* Reset IO tier override to —1 before handling
     syscall */
    movq    TH_TASK(%rcx),%rbx      /* point to current task  */

    /* Check for active vtimers in the current task */
    TASK_VTIMER_CHECK(%rbx,%rcx)

    /*
     * We can be here either for a mach, unix machdep or diag syscall,
     * as indicated by the syscall class:
     */
    movl    R64_RAX(%r15), %eax     /* syscall number/class */
    movl    %eax, %edx
    andl    $(SYSCALL_CLASS_MASK), %edx /* syscall class */
    cmpl    $(SYSCALL_CLASS_MACH<<SYSCALL_CLASS_SHIFT), %edx
    je  EXT(hndl_mach_scall64)
    cmpl    $(SYSCALL_CLASS_UNIX<<SYSCALL_CLASS_SHIFT), %edx
    je  EXT(hndl_unix_scall64)
    cmpl    $(SYSCALL_CLASS_MDEP<<SYSCALL_CLASS_SHIFT), %edx
    je  EXT(hndl_mdep_scall64)
    cmpl    $(SYSCALL_CLASS_DIAG<<SYSCALL_CLASS_SHIFT), %edx
    je  EXT(hndl_diag_scall64)
```

```
39
40      /* Syscall class unknown */
41      sti
42      CCALL3(i386_exception, $(EXC_SYSCALL), %rax, $1)
43      /* no return */
```

Listing 1: 系统调用入口

其中几种系统调用的分类序号为:

```
1 // xnu/osfmk/mach/i386/syscall_sw.h
2 #define SYSCALL_CLASS_NONE 0 /* Invalid */
3 #define SYSCALL_CLASS_MACH 1 /* Mach */
4 #define SYSCALL_CLASS_UNIX 2 /* Unix/BSD */
5 #define SYSCALL_CLASS_MDEP 3 /* Machine-dependent */
6 #define SYSCALL_CLASS_DIAG 4 /* Diagnostics */
7 #define SYSCALL_CLASS_IPC 5  /* Mach IPC */
```

Listing 2: 系统调用分类序号

系统调用号为 *SYSCALL_CLASS « 24 | dispatch_num* 的组合,例如 *SYSCALL_CLASS* 为 3, *dispatch_num* 为 0 则将进入 *hv_task_trap* 并最终进入 AppleHV.kext 对应的处理函数, 示例如下:

```
1 //
2 void trap_by_syscall_ins() {
3     asm ("mov $0x03000000, %rax; mov $0x04, %rdi; syscall")
4 }
```

Listing 3: 使用 syscall 指令

关于 GCC 和 clang 内嵌汇编参考:

- http://cs.du.edu/ mitchell/bootcamp/x86-64.pdf
- https://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html

2 文件系统

3 网络系统

4 内存管理

5 进程管理

6 Mach 相关

7 BSD 相关

8 IOKit

8.1 IOUserClient 对象生命周期

IOUserClient 对象是用户态程序与驱动交互最主要的对象之一，用户态通过 *IOServiceOpen* 函数可以打开一个驱动，返回给用户态的为 *io_connect_t* 类型，对应内核里类型为 **IKOT_IOKIT_CONNECT** 的 port 对象，因而存储在该 port 里的真正对象为成员 kdata.ip_kobject 所对应的 IOUserClient 对象。

```
// device_server.c
/* Routine io_service_open_extended */
void _Xio_service_open_extended(mach_msg_header_t *InHeadP, mach_msg_header_t *OutHeadP)
{
    typedef struct {
        mach_msg_header_t Head;
        /* start of the kernel processed data */
        mach_msg_body_t msgh_body;
        mach_msg_port_descriptor_t owningTask;
        mach_msg_ool_descriptor_t properties;
        /* end of the kernel processed data */
        NDR_record_t NDR;
        uint32_t connect_type;
        NDR_record_t ndr;
        mach_msg_type_number_t propertiesCnt;
        mach_msg_trailer_t trailer;
    } Request __attribute__((unused));

    Request *In0P = (Request *) InHeadP;
    Reply *OutP = (Reply *) OutHeadP;

    kern_return_t RetCode;
    io_object_t service;
    task_t owningTask;
    io_connect_t connection;
    ......
    service = iokit_lookup_object_port(In0P->Head.msgh_request_port);
    owningTask = convert_port_to_task(In0P->owningTask.name);
    RetCode = is_io_service_open_extended(service, owningTask, In0P->connect_type,
                                          In0P->ndr, (io_buf_ptr_t)(In0P->properties.address),
                                          In0P->properties.size, &OutP->result, &connection);
    task_deallocate(owningTask);
    iokit_remove_reference(service);
    if (RetCode != KERN_SUCCESS) {
        MIG_RETURN_ERROR(OutP, RetCode);
    }
    ......
    OutP->connection.name = (mach_port_t)iokit_make_connect_port(connection); <——— (a)
    OutP->NDR = NDR_record;
    OutP->Head.msgh_bits |= MACH_MSGH_BITS_COMPLEX;
    OutP->Head.msgh_size = (mach_msg_size_t)(sizeof(Reply));
    OutP->msgh_body.msgh_descriptor_count = 1;
}
```

Listing 4: _Xio_service_open_extended 函数

在 (a) 处，iokit_make_connect_port 函数将创建一个 IKOT_IOKIT_CONNECT 类型的 mach port，最后调用 iokit_alloc_object_port 函数真正创建并且将 IOUserClient 对象与 port 关联。

```c
// iokit_rpc.c
ipc_port_t iokit_make_connect_port(io_object_t obj) {
    ipc_port_t port;
    ipc_port_t sendPort;

    if (obj == NULL) return IP_NULL;

    port = iokit_port_for_object(obj, IKOT_IOKIT_CONNECT);
    if (port) {
        sendPort = ipc_port_make_send(port);
        iokit_release_port(port);
    } else
        sendPort = IP_NULL;

    iokit_remove_reference(obj);

    return (sendPort);
}
```

Listing 5: iokit_make_connect_port 函数

```c
// iokit_rpc.c
ipc_port_t iokit_alloc_object_port(io_object_t obj, ipc_kobject_type_t type) {
    ipc_port_t notify;
    ipc_port_t port;
    do {
        /* Allocate port, keeping a reference for it. */
        port = ipc_port_alloc_kernel();
        if (port == IP_NULL) continue;
        /* set kobject & type */
        //  iokit_add_reference( obj );
        ipc_kobject_set(port, (ipc_kobject_t)obj, type);
        /* Request no-senders notifications on the port. */
        ip_lock(port);
        notify = ipc_port_make_sonce_locked(port);
        ipc_port_nsrequest(port, 1, notify, &notify);
        /* port unlocked */
        assert(notify == IP_NULL);
        gIOKitPortCount++;
    } while (FALSE);
    return (port);
}
```

Listing 6: iokit_alloc_object_port 函数

在此后一直到内核里的 IOUserClient 对象完全销毁之前，每次通过用户态的 io_connect_t 作为参数的所有 MIG[1] 函数进入内核后都会调用 iokit_lookup_connect_port 进行检查类型检查，并增加 retainCount。

```c
// iokit_rpc.c
io_object_t iokit_lookup_connect_port(ipc_port_t port) {
    io_object_t obj;
    if (!IP_VALID(port)) return (NULL);
    iokit_lock_port(port);
    if (ip_active(port) && (ip_kotype(port) == IKOT_IOKIT_CONNECT)) {
        obj = (io_object_t)port->ip_kobject;
        iokit_add_connect_reference(obj);
    } else
```

[1] http://www.cs.cmu.edu/afs/cs/project/mach/public/doc/unpublished/mig.ps

```
        obj = NULL;
    iokit_unlock_port(port);
    return (obj);
}
```

Listing 7: iokit_lookup_connect_port 函数

9 缓解策略