Linux Firewall Exploration Lab

Xinyi Li

March 28, 2020

Instruction: https://seedsecuritylabs.org/Labs_16.04/PDF/Firewall.pdf

Set up 2 VMs: - A: `10.0.2.15` - B: `10.0.2.4`.

Task 1

All commands run on the VM `10.0.2.15`, and modify Line 11 in its `/etc/default/ufw` with root privilege (e.g. `sudo vi`/`sudo gedit`/`sudo nano`+filename or whatever method you like) as:

```
DEFAULT_INPUT_POLICY="ACCEPT"
```

Or simply use the command:

```
sudo ufw default allow incoming
```

Ref to the manual and tutorial of `ufw`

At first, enable it:

```
sudo ufw enable
```

After configuration, reset it to the installed status and remove added rules:

```
sudo ufw reset
```

Prevent A from doing `telnet` to Machine B

```
sudo ufw deny out from 10.0.2.15 to 10.0.2.4 port 23
```

Prevent B from doing `telnet` to Machine A

```
sudo ufw deny in from 10.0.2.4 to 10.0.2.15 port 23
```

Prevent A from visiting an external web site

Use `ping` or `traceroute` to get one of the host IP address and block the corresponding `HTTP`/`HTTPS` connections:

```
sudo ufw deny out from 10.0.2.15 to host_ip port 80
sudo ufw deny out from 10.0.2.15 to host_ip port 443
```

Note: As this answer explains, It is impossible to stop the accessing to a domain.

Task 2

From the programming manual of `netfilter` module, I can implement a simplified firewall program as `packet_filter.c`

For each rule, a callback function is defined to filter packets meeting some specified conditions.

For example, the function for task 1.1 can be defined as `telnetFilter_1()`

```c
unsigned int telnetFilter_1(void *priv, struct sk_buff *skb,
                            const struct nf_hook_state *state)
// rule for task 1.1: Prevent A from doing `telnet` to Machine B
{
    struct iphdr *iph;
    struct tcphdr *tcph;

    iph = ip_hdr(skb);
    tcph = (void *)iph + iph->ihl * 4;

    if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23)
        && eq_daddr(iph, "10.0.2.4") && eq_saddr(iph,
        "10.0.2.15"))
    {
        printk(KERN_INFO "Dropping telnet from %pI4 packet to
            %pI4\n", &iph->saddr, &iph->daddr);
        return NF_DROP;
    }
    else
    {
        return NF_ACCEPT;
    }
}
```

Similarly, the `if` statement can be replaced by other rules to construct more filters.

The `if`condition in `telnetFilter_2()` for task 1.2:

```c
if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23) &&
    eq_daddr(iph, "10.0.2.15") && eq_saddr(iph, "10.0.2.4"))
```

Assume that we intend to block machine A from opening the website: http://notebook.xyli.me/. Before constructing the rule, we utilize Wireshark to get its 2 host IP address: `104.18.21.226` and `103.235.46.191`.

Based on the knowledge, the `if` condition in 'block_xyli_me' for task 1.3 should be:

```
if ((tcph->dest == htons(80) || tcph->dest == htons(443))
&& (eq_daddr(iph, "104.18.21.226") ||
    eq_daddr(iph,"103.235.46.191"))
&& eq_saddr(iph, "10.0.2.15"))
```

To make it scalable for at least 5 filter rules, a `nf_hook_ops` array should be declared with a large size and `regist_num` is maintained to track the actual amount of filters used in the firewall:

```
#define MAX_RULE_NUM 10

static struct nf_hook_ops FilterHookRule[MAX_RULE_NUM];
static int regist_num = 0;
```

Regist those hooks with functions above in functionsetUpFilter():

```
int setUpFilter(void)
{
    int i;
    printk(KERN_INFO "Registering filters.\n");
    FilterHookRule[0] = (struct nf_hook_ops){.hook =
        telnetFilter_1, .hooknum = NF_INET_LOCAL_OUT, .pf =
        PF_INET, .priority = NF_IP_PRI_FIRST};
    FilterHookRule[1] = (struct nf_hook_ops){.hook =
        telnetFilter_2, .hooknum = NF_INET_LOCAL_IN, .pf =
        PF_INET, .priority = NF_IP_PRI_FIRST};
    FilterHookRule[2] = (struct nf_hook_ops){.hook =
        block_xyli_me, .hooknum = NF_INET_LOCAL_OUT, .pf =
        PF_INET, .priority = NF_IP_PRI_FIRST};

    // set the amount of filter rules
    regist_num = 3;

    for (i = 0; i < regist_num; i++)
        nf_register_hook(&FilterHookRule[i]);
    return 0;
}
```

When extending the module with more rules, just focus to fill out `hooknum` and `hook`(i.e. function definition) fields like this.

Note: `hooknum` field, namely hook type, is not consistent with the identifiers in the book. Please read their actual definition (alias `enum`) in `linux/netfilter.h`.

Define an unregist function as well and associate those functions in the module.

```c
void removeFilter(void)
{
    int i;
    printk(KERN_INFO "Filters are being removed.\n");
    //unregist hooks one by one
    for (i = 0; i < regist_num; i++)
        nf_unregister_hook(&FilterHookRule[i]);
    regist_num = 0;
}

module_init(setUpFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");
```

Finally, write a `makefile` and type `make` to compile:

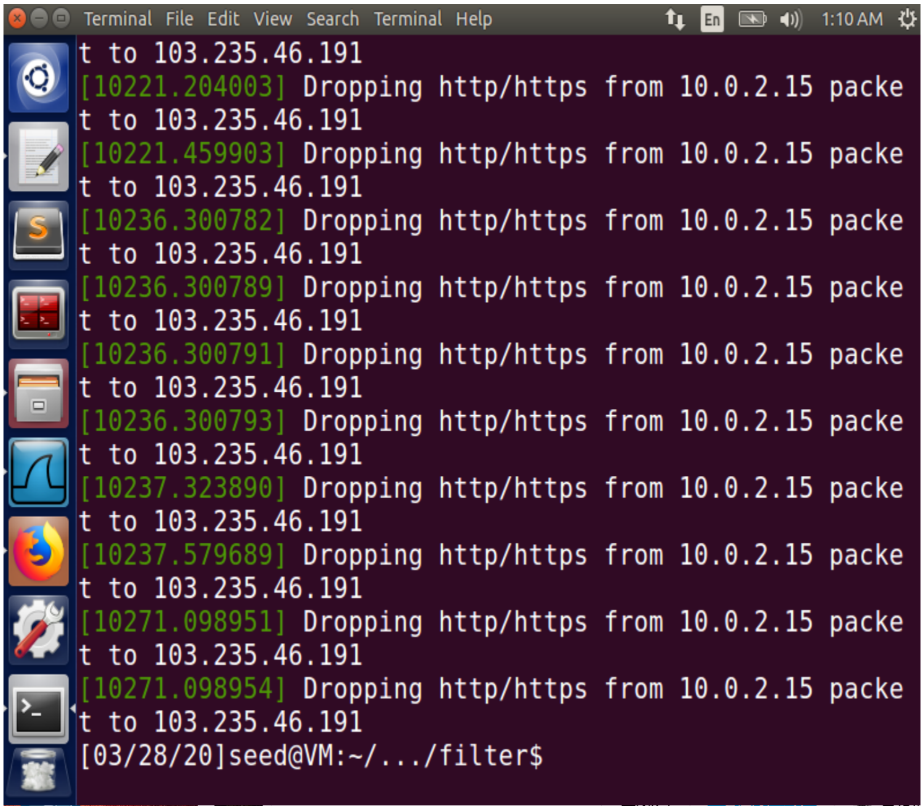
```makefile
obj-m += packet_filter.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Install the output module `packet_filter.ko` into the kernel:

```
sudo insmod packet_filter.ko
```

Now, you can observe the firewall works, and log can be viewed with command `dmseg`:



```
t to 103.235.46.191
[10221.204003] Dropping http/https from 10.0.2.15 packe
t to 103.235.46.191
[10221.459903] Dropping http/https from 10.0.2.15 packe
t to 103.235.46.191
[10236.300782] Dropping http/https from 10.0.2.15 packe
t to 103.235.46.191
[10236.300789] Dropping http/https from 10.0.2.15 packe
t to 103.235.46.191
[10236.300791] Dropping http/https from 10.0.2.15 packe
t to 103.235.46.191
[10236.300793] Dropping http/https from 10.0.2.15 packe
t to 103.235.46.191
[10237.323890] Dropping http/https from 10.0.2.15 packe
t to 103.235.46.191
[10237.579689] Dropping http/https from 10.0.2.15 packe
t to 103.235.46.191
[10271.098951] Dropping http/https from 10.0.2.15 packe
t to 103.235.46.191
[10271.098954] Dropping http/https from 10.0.2.15 packe
t to 103.235.46.191
[03/28/20]seed@VM:~/.../filter$
```

Don't forget to uninstall the firewall after lab:

```
sudo rmmod packet_filter
```