Project: Kernel Module Network Monitor

Member: Chuilian Kong

GithubLink: https://github.com/CS3281-2016/KCL-NetworkTrafficMonitor.git

Kernel Module Project Report

Chuilian Kong

Project Description

In this project I built a kernel module that is capable of monitoring and controlling network traffic. The basic functionalities including “monitor all incoming network traffic” “drop all outgoing network traffic” “drop network traffic from specified address” “set a quota to a destination address”.

Project Design

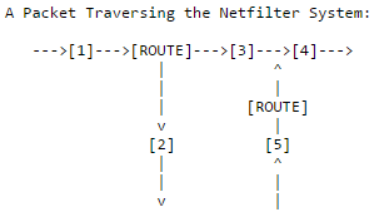
User interface & commands

The user interface of the module is implemented using /proc file system. User can input valid commands to /proc/htmm to enable functionalities of the module. For example, user can turn on incoming monitor by enter “echo “turn on incoming monitor” >/proc/htmm” in the terminal. Valid commands are as follows and the meaning of each command is straightforward (all characters in a command is lower case):

1. Turn on incoming monitor
2. Turn on outgoing monitor
3. Turn on all monitor
4. Turn off incoming monitor
5. Turn off outgoing monitor
6. Turn off all monitor
7. Block incoming
8. Block outgoing
9. Block all
10. Unblock incoming
11. Unblock outgoing
12. Unblock all
13. Block saddr #IP
14. Block daddr #IP
15. Unblock saddr #IP
16. Unblock daddr #IP
17. Clear saddr block list
18. Clear daddr block list
19. View saddr block list
20. View daddr block list
21. Set saddr #IP quota #BYTE
22. Set daddr #IP quota #BYTE
23. Unset quota saddr #IP
24. Unset quota daddr #IP
25. Clear saddr quota list
26. Clear daddr quota list
27. View saddr quota list
28. View daddr quota list
29. View hook table

Implementation & Functionalities

Basically the module will use netfilter hooks to get the control over network traffic.



A incoming network packet will come from the left and pass some of the hooks indicated by numbers, and then enter the local system on the bottom. A outgoing network packet will come from the bottom and pass some of the hooks then leave the system to the right. To implement functionalities of monitoring and controlling the network traffic, what I did is to register some functions on the hook number [2] and hook number [5], and those functions will read the header of every packet walk through it and either ACCEPT it or DROP it depending on information in the header and the functionality we want to implement.

I created four hook functions for four major functionalities of the module and I will describe them here in detail:

-Monitor

Monitor is the most harmless hook function in my design. It simply gather some specific information of the packets walk through it from their header and then let them go by returning NF_ACCEPT. This function will never catch any packet, it just inspects them. The information the monitor function gathered is

1. Source address

2. Destination address

3. Data size of the packet

Actually the monitor will not inspect source address and destination address at the same time. If the monitor is hooked on the NF_INET_LOCAL_IN, it will only inspect source address since the destination must be the local address and if the monitor is hooked on the NF_INET_LOCAL_OUT, it will only inspect destination address. The monitor will print the information it gathered to kernel ring buffer.

-Dropall

Dropall is the most powerful hook function among all. It will drop all network packets that walk through it no matter how harmless the packet seems to be. And it will gather the same information just as the monitor do and print them to kernel ring buffer.

-Dropblocked

Dropblocked seems to be a reasonable hook function. It will maintain a drop list for both source addresses and destination addresses, respectively. User can add entries to either of these lists or delete entries from them. The function will not be hooked until at least one entry is added to the list. For instance, if user add an entry to saddr block list (source address block list), the function will be registered to the NF_INET_LOCAL_IN, if the user keep adding entries to the list, the function will maintain its registration. And if the list is empty, the hook function will be unregistered from the corresponding

position. What the hook function does is basically look on the source/destination address of a packet and then see if it is existed on the saddr/daddr block list, and then make the decision of whether drop or accept. The dropblocked will print the blocked packets info to the kernel ring buffer.

-Quota controller

Quota controller is a more complicated hook function. It will maintain a quota list for both source addresses and destination addresses, respectively. It kind of like the block list except every entry in the quota list including not only an address, but also a quota that the user gives and current traffic that it used. Quota controller will monitor every address on the list, then update their current traffic, then check if the current traffic exceeds the quota, if it does, just put the address in the corresponding block list. The quota controller itself will never drop a packet, the control is enforced by dropblocked function. It will make sure dropblocked function is hooked when needed. The quota controller will print the monitored packets info to the kernel ring buffer including additional info about the quota and current traffic.

About priority

Sometimes user may want to do many jobs at the same time, which is fully supported by my module because I have different hook functions for different purpose and they can be hooked on the same position at the same time. But there is one problem, that is, which function should run first? In my design, the monitor should always run first since the purpose of the monitor function is to see how many packets are initially get in the filter, and then the dropall and dropblocked should run second, and then the quota controller should run last since updating current traffic should be based on actual network traffic (filtered traffic).

By placing these functions on either NF_INET_LOCAL_IN or NF_INET_LOCAL_OUT, we can implement all the functionalities that specified by the commands listed before.

More advanced functionalities: Stateful Firewall

A stateful firewall is a network firewall that tracks the operating state and characteristics of network connections traversing it. The firewall is configured to distinguish legitimate packets for different types of connections. Only packets matching a known active connection are allowed to pass the firewall.

To implement stateful firewall, we first need a data structure to store the information of current network connections, say, a table. In this network connection table, each entry stores information about a connection, and the information includes IP addresses and ports involved in the connection and the sequence numbers of the packets traversing the connection. We then need a new hook function to update that information on the connection table, as well as inspect packets and judge whether they are legitimate, depending on rules involving connection table. For instance, we get a new packet in the hook function, and we see its header, find the source address is on an active connection’s address list, so it is legitimate, we accept it, then we get another packet, and we see its header, find the source address is not on any active connection’s address list, so we drop it. The real rule may be more complicated, but the idea is similar.

Reference

https://www.netfilter.org/documentation/HOWTO/networking-concepts-HOWTO-2.html

http://www.ciscopress.com/articles/article.asp?p=348253&seqNum=4

http://stackoverflow.com/questions/9296835/convert-source-ip-address-from-struct-iphdr-to-string-equivalent-using-linux-ne

http://www.paulkiddie.com/2009/11/creating-a-netfilter-kernel-module-which-filters-udp-packets/

http://lxr.free-electrons.com/

https://linux.die.net/man/

Kernel Module Project Work Highlighting

Chuilian Kong

Material Learning

- Study how to build a kernel module
  - Understand how to use printk
  - Understand the lifecycle of the module
  - Understand how to use dmesg to look up messages from kernel
  - Understand how to use /proc file system to communicate with processes from user space
  - Understand how to allocate and free memory in kernel mode

Material Learning

- Study how to use netfilter with kernel module
  - Study netfilter hooks in existing modules such as iptables
  - Implement the ability to filter all incoming/outgoing traffic
  - Implement the ability to filter all incoming/outgoing traffic from/to specific addresses
  - Create an entry in a log file for each packet that is filtered
  - Implement more advanced functionality such as a stateful firewall

Module Design

- Functionalities:
  - monitor all network traffic on either LOCAL_OUT or LOCAL_IN or both.
  - drop all network traffic on either LOCAL_OUT or LOCAL_IN or both.
  - drop network traffic from specific source address or to specific destination address specified by user, the module will maintain a list for them.
  - monitor network traffic from specific source address or to specific destination address specified by user, and if the network traffic of specified address exceeds the quota set by user, block the incoming/outgoing network for the address.

Module Design

- User Interface
  - /proc/htmm
  - Parse user commands : 29 supported commands
- Hook functions
  - Monitor
  - Dropall
  - Dropblocked
  - Quota controller
- Control Logic
  - Total 1600 lines c code

Test Cases

- Test Basic functionalities
  - Test 22 commands independently
- Test sophisticated scenarios
  - 1 sophisticated test case
- Automatically run test commands and check results
  - Total 700 lines shell script code