LINUX MINIFIREWALL

REPORT

GROUP 13

Aman Agarwal	(15114006)
Anubhav Jain	(15114014)
Devesh Masane	(15114023)
Dhanraj Sahu	(15114024)
Harsh Kumar Bansal	(15114033)

OBJECTIVE:

Our objective is to build a firewall that blocks unauthorized access while permitting authorized communications.Firewalls have several types. In this project, we focus on a very simple type, the packet filter. Packet filters act by inspecting the packets. All datagrams entering or leaving the intranet pass through the firewall, which examines each datagrams and blocks those that do not meet the specified security criteria.

The firewall can BLOCK or UNBLOCK (LOG) packets according to a set of rules. The rule will be added to Packet filters which often use a combination of the packet's source and destination address, its protocol (TCP/UDP) traffic, the port number from a certain source IP network to a destination port will be blocked. .

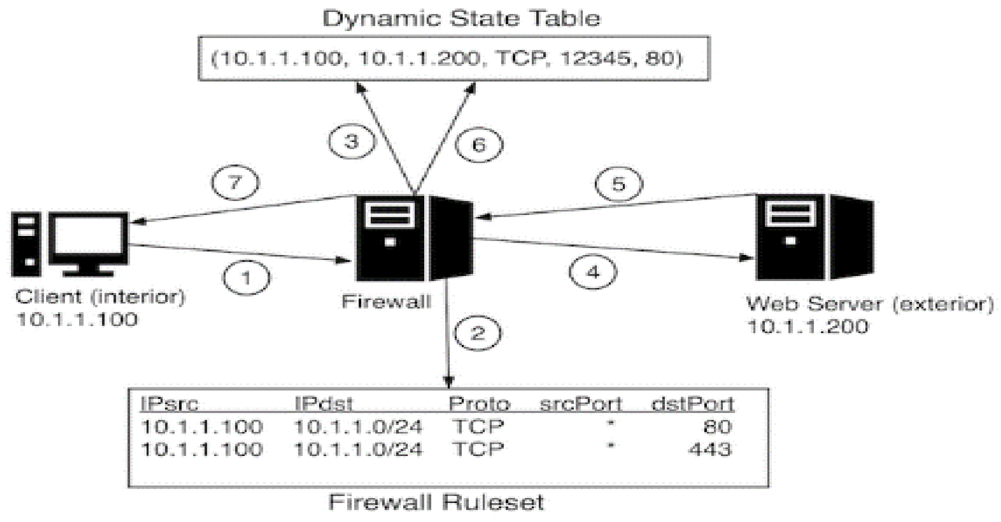

Dynamic State Table
(10.1.1.100, 10.1.1.200, TCP, 12345, 80)
3
6
7
5
1
4
2
Client (interior)
10.1.1.100
Firewall
Web Server (exterior)
10.1.1.200
IPsrc IPdst Proto srcPort dstPort
10.1.1.100 10.1.1.0/24 TCP * 80
10.1.1.100 10.1.1.0/24 TCP * 443
Firewall Ruleset

Figure 1: Interior Client sends a request to an external webserver. (1) Client makes a request to web server at destination IP 10.1.1.200 and destination port 80 using source port 12345, and the TCP protocol. (2) Firewall checks Firewall Ruleset to determine that outgoing connection is permitted. (3) Firewall puts 5-tuple in the Dynamic state table. (4) Firewall permits client request to webserver. (5) Web server responds to client request with reply. (6) Firewall checks the dynamic state table, and finds a matching open connection for the web server reply. (7) Firewall permits webserver reply to reach client.

Packet filtering: It looks at each packet entering or leaving the network and accepts or rejects it based on user-defined laws. Packet filtering is fairly effective and ransparent to users, but it is difficult to configure. In addition, it is susceptible to IP spoofing.

Our project can be divided into two functionalities :
1. Kernel Module
2. UserSpace Program (Configuration Utility)

USERSPACE PROGRAM :

getopt_long() Prototype

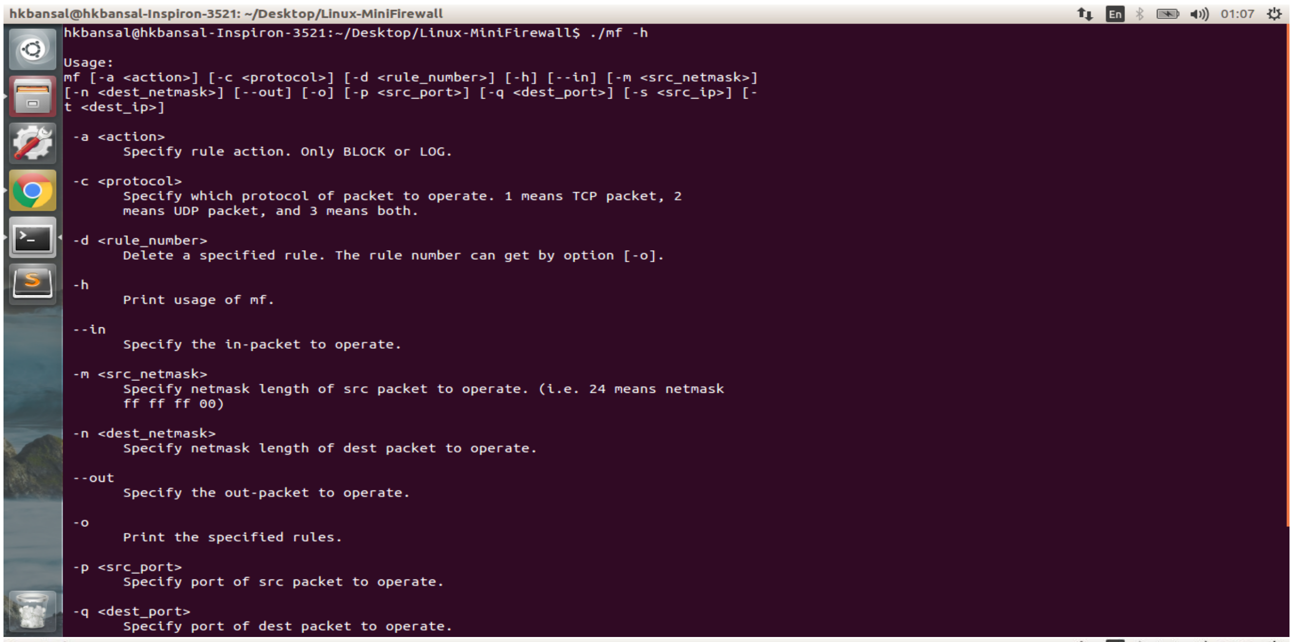
int getopt_long (int argc, char *const *argv, const char *shortopts, const struct option *longopts, int *indexptr)

argc and argv are the argument count and argument vector, the same as the two input parameters in the standard main prototype.

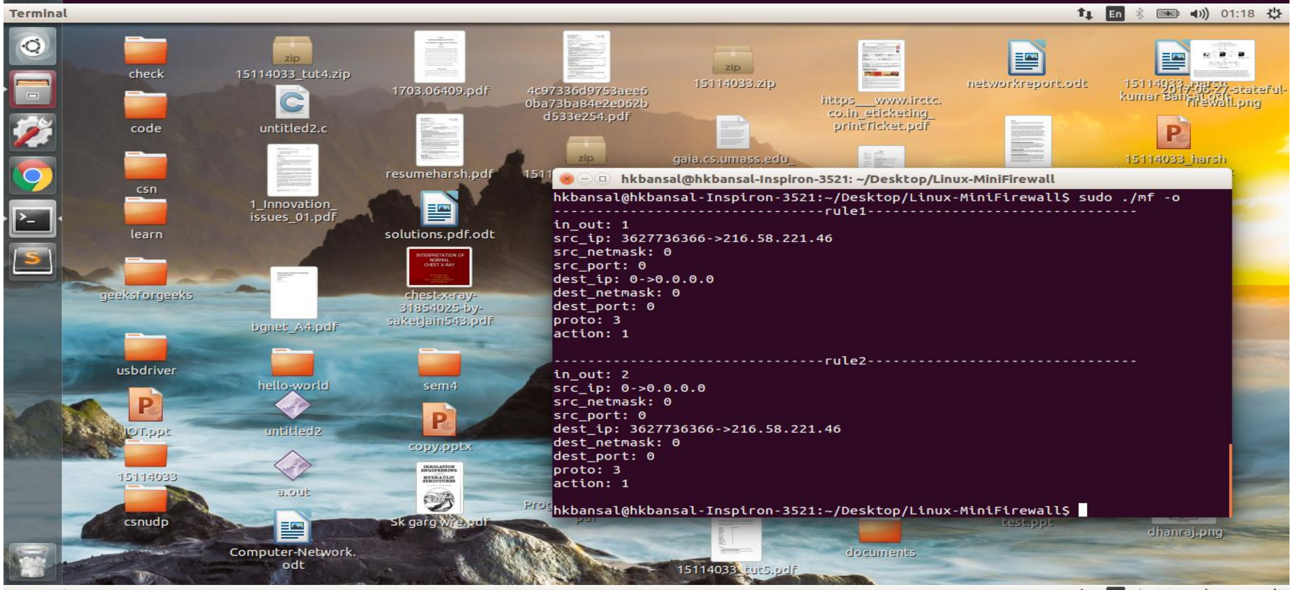
char *short_options = "hod:s:m:p:t:n:q:c:a:";

longopts describes the long options to accept, which is an array of struct option. longopts must be terminated by a struct option of all 0s.

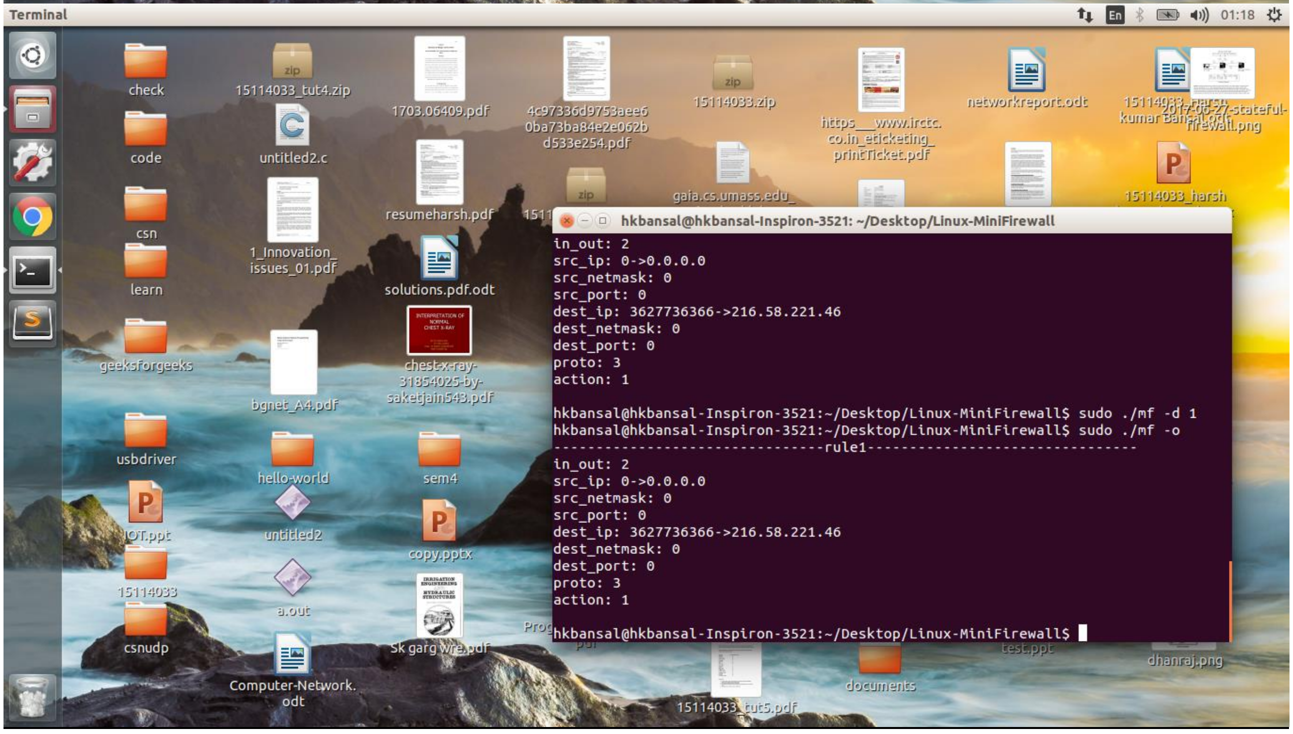
For all options, getopt_long stores the matched option’s index in array longopts to *indexptr. You can access the name of the option by longopts[*indexptr].name.


hkbansal@hkbansal-Inspiron-3521: ~/Desktop/Linux-MiniFirewall
01:07

```
hkbansal@hkbansal-Inspiron-3521:~/Desktop/Linux-MiniFirewall$ ./mf -h

Usage:
mf [-a <action>] [-c <protocol>] [-d <rule_number>] [-h] [--in] [-m <src_netmask>]
[-n <dest_netmask>] [--out] [-o] [-p <src_port>] [-q <dest_port>] [-s <src_ip>] [-
t <dest_ip>]

 -a <action>
       Specify rule action. Only BLOCK or LOG.

 -c <protocol>
       Specify which protocol of packet to operate. 1 means TCP packet, 2
       means UDP packet, and 3 means both.

 -d <rule_number>
       Delete a specified rule. The rule number can get by option [-o].

 -h
       Print usage of mf.

 --in
       Specify the in-packet to operate.

 -m <src_netmask>
       Specify netmask length of src packet to operate. (i.e. 24 means netmask
       ff ff ff 00)

 -n <dest_netmask>
       Specify netmask length of dest packet to operate.

 --out
       Specify the out-packet to operate.

 -o
       Print the specified rules.

 -p <src_port>
       Specify port of src packet to operate.

 -q <dest_port>
       Specify port of dest packet to operate.
```


Terminal
01:18
hkbansal@hkbansal-Inspiron-3521: ~/Desktop/Linux-MiniFirewall

```
hkbansal@hkbansal-Inspiron-3521:~/Desktop/Linux-MiniFirewall$ sudo ./mf -o
--------------------------------rule1--------------------------------
in_out: 1
src_ip: 3627736366->216.58.221.46
src_netmask: 0
src_port: 0
dest_ip: 0->0.0.0.0
dest_netmask: 0
dest_port: 0
proto: 3
action: 1

--------------------------------rule2--------------------------------
in_out: 2
src_ip: 0->0.0.0.0
src_netmask: 0
src_port: 0
dest_ip: 3627736366->216.58.221.46
dest_netmask: 0
dest_port: 0
proto: 3
action: 1

hkbansal@hkbansal-Inspiron-3521:~/Desktop/Linux-MiniFirewall$
```


Terminal
01:18
hkbansal@hkbansal-Inspiron-3521: ~/Desktop/Linux-MiniFirewall

```
in_out: 2
src_ip: 0->0.0.0.0
src_netmask: 0
src_port: 0
dest_ip: 3627736366->216.58.221.46
dest_netmask: 0
dest_port: 0
proto: 3
action: 1

hkbansal@hkbansal-Inspiron-3521:~/Desktop/Linux-MiniFirewall$ sudo ./mf -d 1
hkbansal@hkbansal-Inspiron-3521:~/Desktop/Linux-MiniFirewall$ sudo ./mf -o
--------------------------------rule1--------------------------------
in_out: 2
src_ip: 0->0.0.0.0
src_netmask: 0
src_port: 0
dest_ip: 3627736366->216.58.221.46
dest_netmask: 0
dest_port: 0
proto: 3
action: 1

hkbansal@hkbansal-Inspiron-3521:~/Desktop/Linux-MiniFirewall$
```

KERNEL MODULE :

The set of rules defined by the user utility are implemented in the firewall with the help of a kernel module.
Linux kernel modules are a piece of software that can be loaded/unloaded upon request, either at system start up, or dynamically when the system is up and running. Kernel modules extends the functionalities of the Linux kernel without re-compiling and rebooting the Linux system. Lots of Linux drivers are written as kernel modules and loaded when a compatible device is detected. Linux kernel module exists as .ko file on Linux file system.

INTERNAL STRUCTURE OF KERNEL MODULE :

The rule supplied by procfs virtual file system undergoes through the following functions in kernel module, depending on the arguments given :

- <u>check_rule</u> : this function check any registers with the hooks and check any datagram for the following :
  1. in_out check
  2. protocol check
  3. ip check - it has three conditions
     - ✦ specify ip with it's netmask to block all ip in the net idspecify ip with it's port
     - ✦ specify ip without port
     - ✦ specify ip with it's port
  4. block check

- <u>check_ip</u> : check the two input IP addresses, see if they match, only the first few bits (masked bits) are compared.
- <u>print_a_rule</u>
- <u>delete_a_rule</u>

Apart from this, the kernel module contains
- Creation of a procfs file.
- Creation of a rule structure containing the rule options and a kernel linked list head.
- Memory allocations to the rule structure using '**kmalloc**'.
- Functions to register netfilter hooks with our created kernel functions for rules i.e, hook_in and hook_out functions.
- Ipv4 to string converter.
- Proc file read and write functions.
- Kernel module initialization and cleanup functions.

The output of kernel module can be viewed by the command :
```
sudo tail -f /var/log/messages
```

NETFILTER:

Netfilter is a set of hooks inside Linux kernel. It allows kernel modules to register callback functions with the network stack in order to intercept and manipulate the network packet.

There are five netfilter hooks that programs can register with:

- NF_IP_PRE_ROUTING
- NF_IP_LOCAL_IN
- NF_IP_FORWARD
- NF_IP_LOCAL_OUT
- NF_IP_POST_ROUTING

There are 3 paths through which a packet is traversed:

1. NF_IP_PRE_ROUTING → NF_IP_LOCAL_IN (USED)
2. NF_IP_PRE_ROUTING → NF_IP_FORWARD→ NF_IP_POST_ROUTING
3. NF_IP_LOCAL_OUT→ NF_IP_POST_ROUTING (USED)

Kernel modules can register to any of the 5 hooks along with hook function. When a registered function is called, it can do one of the five things and ***return*** the corresponding value:

- NF_ACCEPT - let the packet pass. (USED)
- NF_DROP - drop the packet. (USED)
- NF_STOLEN - take the packet and don't let the packet pass.
- NF_QUEUE - queue the packet, usually for userspace handling.
- NF_REPEAT - call the hook again.

MEMORY ALLOCATION :

Memory allocation in Linux kernel is different from the user space counterpart. There're two ways to allocate memory space for a kernel process, statically from the stack or dynamically from the heap.

There're two functions available to allocate memory from heap in Linux kernel process

1. vmalloc
2. kmalloc

When a large chunk of memory is needed, vmalloc is used often as it doesn't require physically contiguous memory and the kernel can satisfy the request with much less effort than using kmalloc.

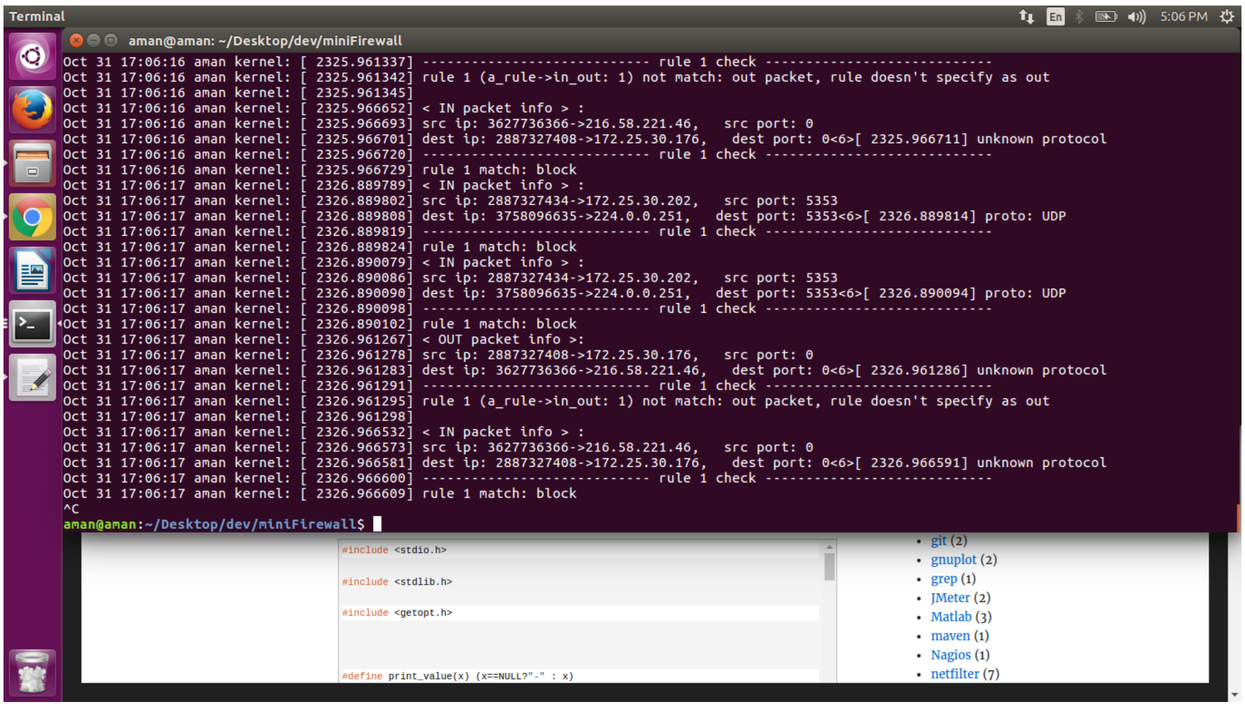
PROCFs :

It can act as a bridge connecting the user space and the kernel space. User space program can use proc files to read the information exported by kernel.Most proc files are read-only and only expose kernel information to user space programs.proc files can also be used to control and modify kernel behavior on the fly.

The registered read callback function is triggered when user space program read the proc file. The write callback function is called when user space program writes to the proc file. The buffer is actually a user space buffer, so it cannot be used directly as the page buffer in proc write function, instead kernel code must use copy_from_user function to get the data and copy_to_user to write in buffer.

Some snapshots of the system log for different types of rules are:

1. When all incoming packets from all IP's are blocked :

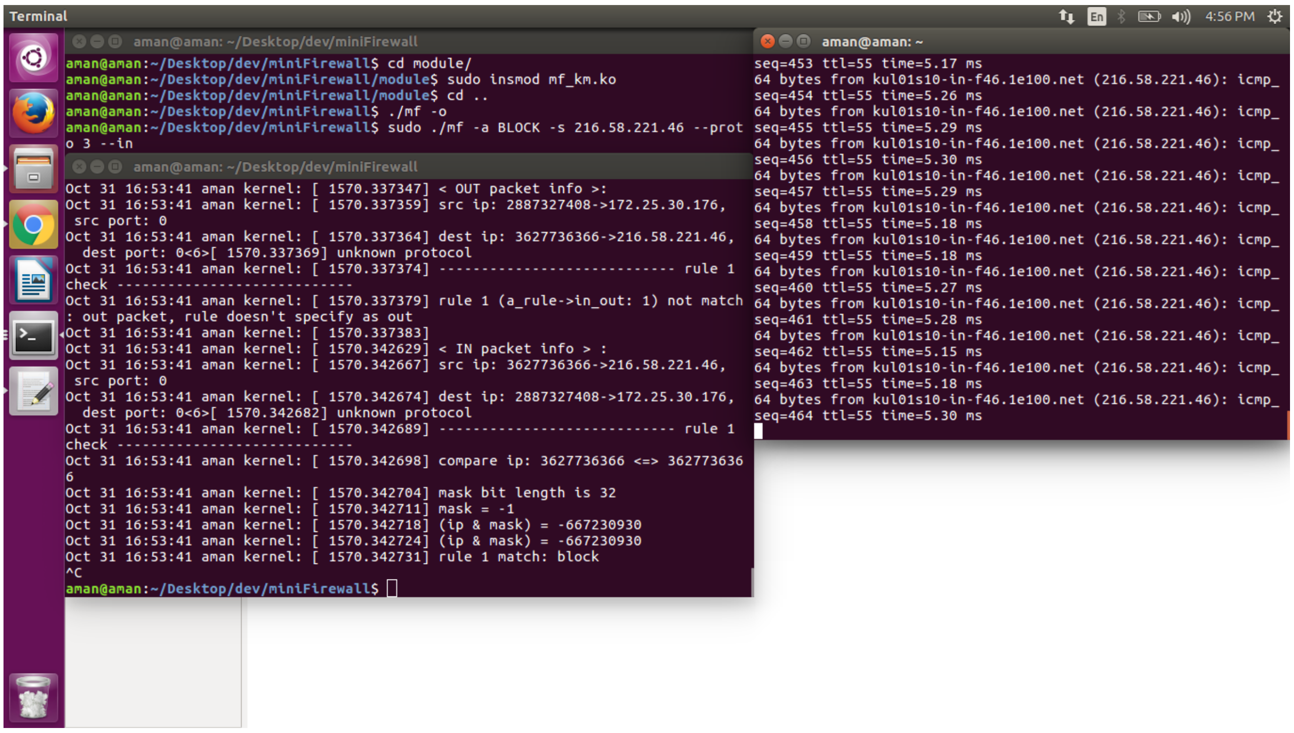


```
Oct 31 17:06:16 aman kernel: [ 2325.961337] ---------------------------- rule 1 check ----------------------------
Oct 31 17:06:16 aman kernel: [ 2325.961342] rule 1 (a_rule->in_out: 1) not match: out packet, rule doesn't specify as out
Oct 31 17:06:16 aman kernel: [ 2325.961345]
Oct 31 17:06:16 aman kernel: [ 2325.966652] < IN packet info > :
Oct 31 17:06:16 aman kernel: [ 2325.966693] src ip: 3627736366->216.58.221.46,   src port: 0
Oct 31 17:06:16 aman kernel: [ 2325.966701] dest ip: 2887327408->172.25.30.176,   dest port: 0<6>[ 2325.966711] unknown protocol
Oct 31 17:06:16 aman kernel: [ 2325.966720] ---------------------------- rule 1 check ----------------------------
Oct 31 17:06:16 aman kernel: [ 2325.966729] rule 1 match: block
Oct 31 17:06:17 aman kernel: [ 2326.889789] < IN packet info > :
Oct 31 17:06:17 aman kernel: [ 2326.889802] src ip: 2887327434->172.25.30.202,   src port: 5353
Oct 31 17:06:17 aman kernel: [ 2326.889808] dest ip: 3758096635->224.0.0.251,   dest port: 5353<6>[ 2326.889814] proto: UDP
Oct 31 17:06:17 aman kernel: [ 2326.889819] ---------------------------- rule 1 check ----------------------------
Oct 31 17:06:17 aman kernel: [ 2326.889824] rule 1 match: block
Oct 31 17:06:17 aman kernel: [ 2326.890079] < IN packet info > :
Oct 31 17:06:17 aman kernel: [ 2326.890086] src ip: 2887327434->172.25.30.202,   src port: 5353
Oct 31 17:06:17 aman kernel: [ 2326.890090] dest ip: 3758096635->224.0.0.251,   dest port: 5353<6>[ 2326.890094] proto: UDP
Oct 31 17:06:17 aman kernel: [ 2326.890098] ---------------------------- rule 1 check ----------------------------
Oct 31 17:06:17 aman kernel: [ 2326.890102] rule 1 match: block
Oct 31 17:06:17 aman kernel: [ 2326.961267] < OUT packet info >:
Oct 31 17:06:17 aman kernel: [ 2326.961278] src ip: 2887327408->172.25.30.176,   src port: 0
Oct 31 17:06:17 aman kernel: [ 2326.961283] dest ip: 3627736366->216.58.221.46,   dest port: 0<6>[ 2326.961286] unknown protocol
Oct 31 17:06:17 aman kernel: [ 2326.961291] ---------------------------- rule 1 check ----------------------------
Oct 31 17:06:17 aman kernel: [ 2326.961295] rule 1 (a_rule->in_out: 1) not match: out packet, rule doesn't specify as out
Oct 31 17:06:17 aman kernel: [ 2326.961298]
Oct 31 17:06:17 aman kernel: [ 2326.966532] < IN packet info > :
Oct 31 17:06:17 aman kernel: [ 2326.966573] src ip: 3627736366->216.58.221.46,   src port: 0
Oct 31 17:06:17 aman kernel: [ 2326.966581] dest ip: 2887327408->172.25.30.176,   dest port: 0<6>[ 2326.966591] unknown protocol
Oct 31 17:06:17 aman kernel: [ 2326.966600] ---------------------------- rule 1 check ----------------------------
Oct 31 17:06:17 aman kernel: [ 2326.966609] rule 1 match: block
^C
aman@aman:~/Desktop/dev/miniFirewall$
```

```
#include <stdio.h>

#include <stdlib.h>

#include <getopt.h>

#define print_value(x) (x==NULL?"-" : x)
```

- git (2)
- gnuplot (2)
- grep (1)
- JMeter (2)
- Matlab (3)
- maven (1)
- Nagios (1)
- netfilter (7)

2. When incoming packets from a particular IP (www.youtube.com- 216.58.221.46) are blocked :

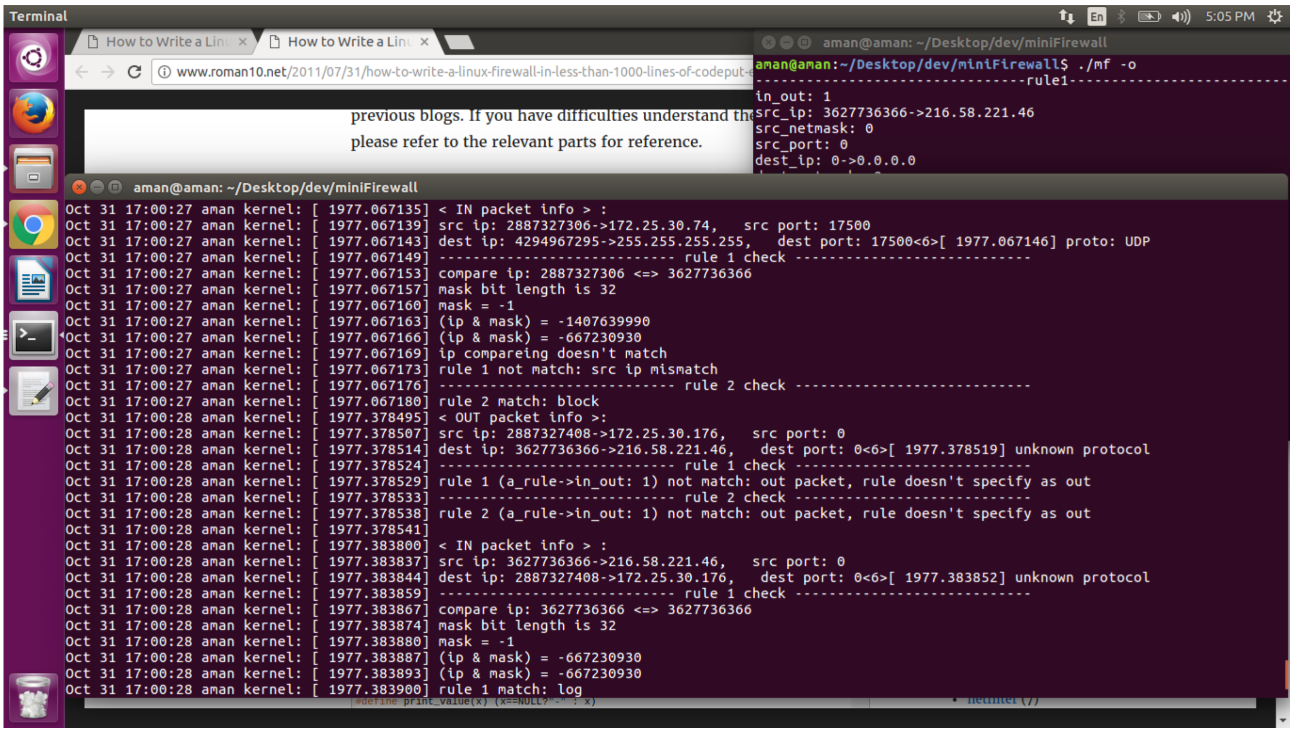


```
aman@aman:~/Desktop/dev/miniFirewall$ cd module/
aman@aman:~/Desktop/dev/miniFirewall/module$ sudo insmod mf_km.ko
aman@aman:~/Desktop/dev/miniFirewall/module$ cd ..
aman@aman:~/Desktop/dev/miniFirewall$ ./mf -o
aman@aman:~/Desktop/dev/miniFirewall$ sudo ./mf -a BLOCK -s 216.58.221.46 --prot
o 3 --in
```

```
Oct 31 16:53:41 aman kernel: [ 1570.337347] < OUT packet info >:
Oct 31 16:53:41 aman kernel: [ 1570.337359] src ip: 2887327408->172.25.30.176,
 src port: 0
Oct 31 16:53:41 aman kernel: [ 1570.337364] dest ip: 3627736366->216.58.221.46,
  dest port: 0<6>[ 1570.337369] unknown protocol
Oct 31 16:53:41 aman kernel: [ 1570.337374] ---------------------------- rule 1
check ----------------------------
Oct 31 16:53:41 aman kernel: [ 1570.337379] rule 1 (a_rule->in_out: 1) not match
: out packet, rule doesn't specify as out
Oct 31 16:53:41 aman kernel: [ 1570.337383]
Oct 31 16:53:41 aman kernel: [ 1570.342629] < IN packet info > :
Oct 31 16:53:41 aman kernel: [ 1570.342667] src ip: 3627736366->216.58.221.46,
 src port: 0
Oct 31 16:53:41 aman kernel: [ 1570.342674] dest ip: 2887327408->172.25.30.176,
  dest port: 0<6>[ 1570.342682] unknown protocol
Oct 31 16:53:41 aman kernel: [ 1570.342689] ---------------------------- rule 1
check ----------------------------
Oct 31 16:53:41 aman kernel: [ 1570.342698] compare ip: 3627736366 <=> 362773636
6
Oct 31 16:53:41 aman kernel: [ 1570.342704] mask bit length is 32
Oct 31 16:53:41 aman kernel: [ 1570.342711] mask = -1
Oct 31 16:53:41 aman kernel: [ 1570.342718] (ip & mask) = -667230930
Oct 31 16:53:41 aman kernel: [ 1570.342724] (ip & mask) = -667230930
Oct 31 16:53:41 aman kernel: [ 1570.342731] rule 1 match: block
^C
aman@aman:~/Desktop/dev/miniFirewall$
```

```
seq=453 ttl=55 time=5.17 ms
64 bytes from kul01s10-in-f46.1e100.net (216.58.221.46): icmp_
seq=454 ttl=55 time=5.26 ms
64 bytes from kul01s10-in-f46.1e100.net (216.58.221.46): icmp_
seq=455 ttl=55 time=5.29 ms
64 bytes from kul01s10-in-f46.1e100.net (216.58.221.46): icmp_
seq=456 ttl=55 time=5.30 ms
64 bytes from kul01s10-in-f46.1e100.net (216.58.221.46): icmp_
seq=457 ttl=55 time=5.29 ms
64 bytes from kul01s10-in-f46.1e100.net (216.58.221.46): icmp_
seq=458 ttl=55 time=5.18 ms
64 bytes from kul01s10-in-f46.1e100.net (216.58.221.46): icmp_
seq=459 ttl=55 time=5.18 ms
64 bytes from kul01s10-in-f46.1e100.net (216.58.221.46): icmp_
seq=460 ttl=55 time=5.27 ms
64 bytes from kul01s10-in-f46.1e100.net (216.58.221.46): icmp_
seq=461 ttl=55 time=5.28 ms
64 bytes from kul01s10-in-f46.1e100.net (216.58.221.46): icmp_
seq=462 ttl=55 time=5.15 ms
64 bytes from kul01s10-in-f46.1e100.net (216.58.221.46): icmp_
seq=463 ttl=55 time=5.18 ms
64 bytes from kul01s10-in-f46.1e100.net (216.58.221.46): icmp_
seq=464 ttl=55 time=5.30 ms
```

3. When incoming packets from all IP’s except 216.58.221.46 are blocked :



```
aman@aman:~/Desktop/dev/miniFirewall$ ./mf -o
--------------------------------rule1--------------------------
in_out: 1
src_ip: 3627736366->216.58.221.46
src_netmask: 0
src_port: 0
dest_ip: 0->0.0.0.0
```

```
Oct 31 17:00:27 aman kernel: [ 1977.067135] < IN packet info > :
Oct 31 17:00:27 aman kernel: [ 1977.067139] src ip: 2887327306->172.25.30.74,   src port: 17500
Oct 31 17:00:27 aman kernel: [ 1977.067143] dest ip: 4294967295->255.255.255.255,   dest port: 17500<6>[ 1977.067146] proto: UDP
Oct 31 17:00:27 aman kernel: [ 1977.067149] ---------------------------- rule 1 check ----------------------------
Oct 31 17:00:27 aman kernel: [ 1977.067153] compare ip: 2887327306 <=> 3627736366
Oct 31 17:00:27 aman kernel: [ 1977.067157] mask bit length is 32
Oct 31 17:00:27 aman kernel: [ 1977.067160] mask = -1
Oct 31 17:00:27 aman kernel: [ 1977.067163] (ip & mask) = -1407639990
Oct 31 17:00:27 aman kernel: [ 1977.067166] (ip & mask) = -667230930
Oct 31 17:00:27 aman kernel: [ 1977.067169] ip compareing doesn't match
Oct 31 17:00:27 aman kernel: [ 1977.067173] rule 1 not match: src ip mismatch
Oct 31 17:00:27 aman kernel: [ 1977.067176] ---------------------------- rule 2 check ----------------------------
Oct 31 17:00:27 aman kernel: [ 1977.067180] rule 2 match: block
Oct 31 17:00:28 aman kernel: [ 1977.378495] < OUT packet info >:
Oct 31 17:00:28 aman kernel: [ 1977.378507] src ip: 2887327408->172.25.30.176,   src port: 0
Oct 31 17:00:28 aman kernel: [ 1977.378514] dest ip: 3627736366->216.58.221.46,   dest port: 0<6>[ 1977.378519] unknown protocol
Oct 31 17:00:28 aman kernel: [ 1977.378524] ---------------------------- rule 1 check ----------------------------
Oct 31 17:00:28 aman kernel: [ 1977.378529] rule 1 (a_rule->in_out: 1) not match: out packet, rule doesn't specify as out
Oct 31 17:00:28 aman kernel: [ 1977.378533] ---------------------------- rule 2 check ----------------------------
Oct 31 17:00:28 aman kernel: [ 1977.378538] rule 2 (a_rule->in_out: 1) not match: out packet, rule doesn't specify as out
Oct 31 17:00:28 aman kernel: [ 1977.378541]
Oct 31 17:00:28 aman kernel: [ 1977.383800] < IN packet info > :
Oct 31 17:00:28 aman kernel: [ 1977.383837] src ip: 3627736366->216.58.221.46,   src port: 0
Oct 31 17:00:28 aman kernel: [ 1977.383844] dest ip: 2887327408->172.25.30.176,   dest port: 0<6>[ 1977.383852] unknown protocol
Oct 31 17:00:28 aman kernel: [ 1977.383859] ---------------------------- rule 1 check ----------------------------
Oct 31 17:00:28 aman kernel: [ 1977.383867] compare ip: 3627736366 <=> 3627736366
Oct 31 17:00:28 aman kernel: [ 1977.383874] mask bit length is 32
Oct 31 17:00:28 aman kernel: [ 1977.383880] mask = -1
Oct 31 17:00:28 aman kernel: [ 1977.383887] (ip & mask) = -667230930
Oct 31 17:00:28 aman kernel: [ 1977.383893] (ip & mask) = -667230930
Oct 31 17:00:28 aman kernel: [ 1977.383900] rule 1 match: log
```

CONTRIBUTION :

USERSPACE PROGRAM :
Harsh Kumar Bansal (15114033)
Dhanraj Sahu(15114024)

KERNEL MODULE :
Anubhav Jain (15114014)
Devesh Masane (15114023)
Aman Agarwal (15114006)