Modules:

1. **struct:** It is used for conversion between C structs and python values. It can handle binary data from network connection or from a file.
2. **sys:** It provides access to variables and functions that interact with the interpreter.
3. **time:** Time related functionalities are provided.
4. **socket:** BSD socket interface can be accessed using this module.
5. **nfqueue:** Used to access packets matched by rules configured in iptables in Linux OS. Various action can be taken on such packets such as accept, drop or alter.
6. **scapy:** It is an interactive packet manipulation program. It can decode packets of various protocols capture, forge and send them on wire. Scapy supports tracerouting, scanning, probing, attacks and network discovery.
7. **math:** It provides access for C defined mathematical functions.
8. **re:** Regular matching expressions can be performed using this module. Pattern and strings can be Unicode and 8-bit string to be searched.
9. **binascii:** provides method to convert binary to various ASCII-encoded binary representations and vice-versa.

Code Explanation:

Import modules used to implement firewall mechanism using packet manipulation.

```
import struct
import sys
import time
import socket
from socket import AF_INET, AF_INET6, inet_ntoa
import nfqueue
from scapy.all import send, IP, TCP

#from main import PKT_DIR_INCOMING, PKT_DIR_OUTGOING
from math import ceil
import re
import binascii
```

Define global variables. __init__ , a constructor method is defined for class Firewall. The argument self is a variable that points to the current object.

```
global action, inputprotocol, actiontype, inputportnum, inputipaddr

class Firewall:
    def __init__(self):
        self.protocolname = ''
        self.srcipaddress = ''
        self.srcportnum = 0
```

This function takes packet as an argument and checks for a valid ip address. It returns true if found valid else returns false.

This function takes packet as an argument. It extracts and returns details of protocol and total length field from the packet. It prints the error and returns None for both the fields if error is occured.

```
def obtain_fields(self, pckt):
    try:
        protocol = struct.unpack('!B', pckt[9:10]) # (integer,)
        total_length = struct.unpack('!H', pckt[2:4])
        return self.strip_format(protocol), self.strip_format(total_length)
    except struct.error as e:
        print (e)
        return None, None
```

This function takes packet as an argument and checks for a valid ip header. It returns true if found valid else prints the error and returns false.

```
def valid_ip_header(self, pckt):
    try:
        #print pckt
        ip_header = struct.unpack('!B', pckt[0:1])
        return self.strip_format(ip_header)
    except struct.error as e:
        print (e)
        print pckt[0:1]
        return None
```

This function takes packet as an argument. It extracts and returns length of UDP header length from the packet. It returns if error is occured.

```
def get_udp_length(self, pckt, startIndex):
    try:
        length = struct.unpack('!H', pckt[startIndex + 4 : startIndex + 6])
        return self.strip_format(length)
    except struct.error:
        return None
```

Packet handling is done in this function. It takes packet and direction of the packet as input. First it checks for valid ip header by calling the function valid_ip_header. If the ip header is correct then the protocol and total length field details are obtained by calling obtain_fields function. Validations on the protocol and total length are performed.

Source and destination address are extracted from the packet. A variable external_addr is initialized to source or destination address based on the direction of the packet.

```
src_addr, dst_addr, pckt_dir = pckt[12:16], pckt[16:20], self.packet_direction(pckt_dir)
if (pckt_dir == 'incoming'):
    external_addr = src_addr
else:
    external_addr = dst_addr
if not (self.valid_IP_address(external_addr)): # check valid address.
    print (6)
    return
```

If the protocol field is ‘6’, then its a TCP packet. A function handle_external_port is called to get port number from the packet and store it in variable external_port. Proper arguments are passed to the function based on the direction of the packet.

```
if (protocol == 6): # TCP
    if (pckt_dir == 'incoming'):
        external_port = self.handle_external_port(pckt, (ip_header) * 4)
    else:
        external_port = self.handle_external_port(pckt, ((ip_header) * 4) + 2)
    if (external_port == None): # drop packet due to port socket error.
        print (7)
        return
```

Similarly, if the protocol field is ‘1’, it’s a ICMP packet. A function handle_icmp_packet is called to handle ICMP packet. It returns a type field.

```
elif (protocol == 1): # ICMP
    type_field = self.handle_icmp_packet(pckt, (ip_header * 4))
    if (type_field == None):
        print (8)
        return
```

If the protocol field is ‘17’, then its a UDP packet. A function handle_external_port is called to get port number from the packet and store it in variable external_port. Proper arguments are passed to the function based on the direction of the packet.

```
elif (protocol == 17): # UDP

    udp_length = self.get_udp_length(pckt, (ip_header * 4))
    if (udp_length == None or udp_length < 8):
        print (9)
        return
    if (pckt_dir == 'incoming'):
        external_port = self.handle_external_port(pckt, (ip_header) * 4)
        if (external_port == None):
            print (10)
            return
    else:
        external_port = self.handle_external_port(pckt, ((ip_header) * 4) + 2)
        if (external_port == None):
            print (10)
            return
```

The verdict variable is set to pass if the protocol is TCP, UDP or ICMP. Protocol name, externl_addr and external_port details obtained from the above functions are stored in the instantaneous object for further processing of packet.

```
verdict = "pass"
self.protocolname = self.protocol_selector(protocol)
self.srcipaddress = external_addr
if (protocol != 1):
    self.srcportnum = external_port
```

This function takes protocol as input and returns corresponding protocol name. It returns None if the protocol is not TCP, UDP or ICMP.

```
""" Protocol Selector."""
def protocol_selector(self, protocol):
    if (protocol == 1):
        return "icmp"
    elif (protocol == 6):
        return 'tcp'
    elif (protocol == 17):
        return 'udp'
    return None
```

- check_protocol function returns true if the protocol is TCP, UDP or ICMP else returns false.
- within_range function returns true of the ip address provided is within range of the start_port and end_port.
- _is_IP_Prefix function checks if the data is an IP prefix.

strip_format function takes argument as a string, removes comma and parentheses and converts into int.

```
""" Strips the parentheses and comma off the number and converts string to int."""
def strip_format(self, format_str):
    new_str = str(format_str)
    return int(new_str[1: len(new_str) - 2])
```

handle_external_port function returns external port. It checks for socket error and if exit, None is returned.

```
""" Returns the external port and checks to see if there is a socket error. If
    the port is valid, then it returns a number, else it returns 'None'. """
def handle_external_port(self, pckt, startIndex):
    try:
        ext_port = pckt[startIndex : startIndex + 2]
        ext_port = struct.unpack('!H', ext_port)
        return ext_port
    except struct.error:
        return None
```

handle_icmp_packet function returns type field using arguments as packet and startIndex. Returns None if there error is occured.

```
""" Returns the TYPE field for the IMCP packet."""
def handle_icmp_packet(self, pckt, startIndex):
    try:
        type_field = pckt[startIndex : startIndex + 1]
        type_field = struct.unpack('!B', type_field)
        return self.strip_format(type_field)
    except struct.error:
        return None
```

packet_direction function returns direction of the packet.

```
""" Returns the direction of the packet in a string."""
def packet_direction(self, direction):
    if (direction == 'outgoing'):
        return 'outgoing'
    else:
        return 'incoming'
```

Packet handling is defined in cb function based on the actions defined in the main function. Incoming packet can be either block or accept. action type can be protocol, IP address or port number. Based on the actiontype provided in main function, the packets are dropped by setting verdict to nfqueue.NF_DROP. Corresponding message is displayed based on action type. If the action is accept then the verdict set to nfqueue.NF_ACCEPT and **Packet accept** message is displayed.

```
def cb(i, payload):
    data = payload.get_data()
    pkt = IP(data)

    f = Firewall()
    f.handle_packet("incoming", str(pkt))
    print (f.protocolname)
    print (socket.inet_ntoa(f.srcipaddress))
    print (f.srcportnum)

    if action == "block":
        if actiontype == "protocol":
            if inputprotocol == f.protocolname:  #TCP, UDP, ICMP
                payload.set_verdict(nfqueue.NF_DROP)
                print (f.protocolname + " Packet blocked")

        elif actiontype == "ipaddress":
            if inputipaddr == socket.inet_ntoa(f.srcipaddress):  #IP address
                payload.set_verdict(nfqueue.NF_DROP)
                print (socket.inet_ntoa(f.srcipaddress) + " blocked")

        elif actiontype == "portnum":
            if int(inputportnum) in f.srcportnum:
                payload.set_verdict(nfqueue.NF_DROP)
                print (inputportnum + " blocked")

    elif action == "accept":
        payload.set_verdict(nfqueue.NF_ACCEPT)
        print ("Packet accepted")

    else:
        payload.set_verdict(nfqueue.NF_ACCEPT)
        print ("Packet accepted")
```

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "ipaddress"
    action = "accept"
    inputportnum = ""

    q = nfqueue.queue()
    q.open()
    q.bind(socket.AF_INET)
    q.set_callback(cb)
    q.create_queue(0)

    try:
        q.try_run()
    except KeyboardInterrupt, e:
        print ("Interrupted")

    print ("unbind")
    q.unbind(AF_INET)
    print ("close")
    q.close()

main()
```

main function defines the firewall rules in the form of action and action type. queue 0 is assigned to nfqueue. All packets can enter from queue 0 and function cb is called to handle the packet

Test Results

a) Protocol

- Accept ICMP packets

Code changes :

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "protocol"
    inputprotocol = "icmp"
    action = "accept"
```

Result:

Ping www.google.com

```
root@tejas-Inspiron-5520:/home/tejas# ping www.google.com
PING www.google.com (172.217.11.164) 56(84) bytes of data.
64 bytes from lax28s15-in-f4.1e100.net (172.217.11.164): icmp_seq=1 ttl=53 time=24.1 ms
64 bytes from lax28s15-in-f4.1e100.net (172.217.11.164): icmp_seq=2 ttl=53 time=64.7 ms
^C
--- www.google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 24.112/44.442/64.772/20.330 ms
```

```
root@tejas-Inspiron-5520:/home/tejas/PY# python UbuntuFirewall_Sample.py
icmp
172.217.11.164
0
Packet accepted
icmp
172.217.11.164
0
Packet accepted
^C
^CInterrupted
unbind
close
```

- Block ICMP packets

Code Changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "protocol"
    inputprotocol = "icmp"
    action = "block"
```

Result:

Ping www.google.com

```
root@tejas-Inspiron-5520:/home/tejas# ping www.google.com
PING www.google.com (172.217.11.164) 56(84) bytes of data.
^C
--- www.google.com ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1017ms
```

```
root@tejas-Inspiron-5520:/home/tejas/PY# python UbuntuFirewall_Sample.py
icmp
172.217.11.164
0
icmp Packet blocked
icmp
172.217.11.164
0
icmp Packet blocked
```

- Accept TCP packets

Code changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "protocol"
    inputprotocol = "tcp"
    action = "accept"
```

Result:

```
root@tejas-Inspiron-5520:/home/tejas# nping -c 1 --tcp -p 80 www.google.com

Starting Nping 0.7.01 ( https://nmap.org/nping ) at 2017-11-20 12:33 PST
SENT (0.1471s) TCP 10.0.0.210:29163 > 172.217.11.164:80 S ttl=64 id=19691 iplen=40  seq=1682947375 win=1480
RCVD (0.3215s) TCP 172.217.11.164:80 > 10.0.0.210:29163 SA ttl=56 id=41129 iplen=44  seq=508177100 win=42780 <mss 1380>

Max rtt: 174.332ms | Min rtt: 174.332ms | Avg rtt: 174.332ms
Raw packets sent: 1 (40B) | Rcvd: 1 (44B) | Lost: 0 (0.00%)
Nping done: 1 IP address pinged in 1.18 seconds
```

```
root@tejas-Inspiron-5520:/home/tejas/PY# python UbuntuFirewall_Sample.py
tcp
172.217.11.164
(80,)
Packet accepted
^CInterrupted
unbind
close
```

- Block TCP packets

Code changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "protocol"
    inputprotocol = "tcp"
    action = "block"
```

Result:

```
root@tejas-Inspiron-5520:/home/tejas# nping -c 1 --tcp -p 80 www.google.com

Starting Nping 0.7.01 ( https://nmap.org/nping ) at 2017-11-20 12:51 PST
SENT (0.0962s) TCP 10.0.0.210:5836 > 216.58.216.36:80 S ttl=64 id=1073 iplen=40  seq=4133652125 win=1480
RCVD (0.2823s) TCP 216.58.216.36:80 > 10.0.0.210:5836 SA ttl=56 id=6509 iplen=44  seq=599270684 win=42780 <mss 1380>
RCVD (0.6904s) TCP 216.58.216.36:80 > 10.0.0.210:5836 SA ttl=56 id=6562 iplen=44  seq=599270684 win=42780 <mss 1380>

Max rtt: 594.116ms | Min rtt: 186.045ms | Avg rtt: 390.080ms
Raw packets sent: 1 (40B) | Rcvd: 2 (88B) | Lost: 0 (0.00%)
Nping done: 1 IP address pinged in 1.13 seconds
```

```
root@tejas-Inspiron-5520:/home/tejas/PY# python UbuntuFirewall_Sample.py
tcp
216.58.216.36
(80,)
tcp Packet blocked
tcp
216.58.216.36
(80,)
tcp Packet blocked
^CInterrupted
unbind
close
```

- Accept UDP packets

Code changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "protocol"
    inputprotocol = "udp"
    action = "accept"
```

Result:

```
root@tejas-Inspiron-5520:/home/tejas/PY# python UbuntuFirewall_Sample.py
udp
127.0.0.1
(25067,)
Packet accepted
udp
127.0.0.1
(25067,)
Packet accepted
udp
127.0.0.1
(25067,)
Packet accepted
udp
127.0.0.1
(25067,)
Packet accepted
udp
127.0.0.1
(25067,)
Packet accepted
udp
127.0.0.1
(25067,)
Packet accepted
^CInterrupted
unbind
close
```

- Block UDP packets

Code changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "protocol"
    inputprotocol = "udp"
    action = "block"
```

Results:

```
root@tejas-Inspiron-5520:/home/tejas/PY# python UbuntuFirewall_Sample.py
udp
127.0.0.1
(25067,)
udp Packet blocked
udp
127.0.0.1
(25067,)
udp Packet blocked
udp
127.0.0.1
(25067,)
udp Packet blocked
^CInterrupted
unbind
close
```

b) IP address

- Accept packets from 172.217.11.164

Code changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "ipaddress"
    inputipaddr = "172.217.11.164"
    action = "accept"
```

```
root@tejas-Inspiron-5520:/home/tejas/PY# python UbuntuFirewall_Sample.py
tcp
172.217.11.164
(80,)
Packet accepted
tcp
172.217.11.164
(80,)
Packet accepted
tcp
172.217.11.164
(80,)
Packet accepted
^CInterrupted
unbind
close
```

Result:

- Block packets from 172.217.11.164

Code changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "ipaddress"
    inputipaddr = "172.217.11.164"
    action = "block"
```

Result:

```
root@tejas-Inspiron-5520:/home/tejas/PY# python UbuntuFirewall_Sample.py
tcp
172.217.11.164
(80,)
172.217.11.164 blocked
tcp
172.217.11.164
(80,)
172.217.11.164 blocked
tcp
172.217.11.164
(80,)
172.217.11.164 blocked
^CInterrupted
unbind
close
```

- Accept packets from 127.0.0.1

Code changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "ipaddress"
    inputipaddr = "127.0.0.1"
    action = "accept"
```

Result:

```
root@tejas-Inspiron-5520:/home/tejas/PY# python UbuntuFirewall_Sample.py
udp
127.0.0.1
(25067,)
Packet accepted
udp
127.0.0.1
(25067,)
Packet accepted
udp
127.0.0.1
(25067,)
Packet accepted
udp
127.0.0.1
(25067,)
Packet accepted
udp
127.0.0.1
(25067,)
Packet accepted
udp
127.0.0.1
(25067,)
Packet accepted
udp
127.0.0.1
(25067,)
Packet accepted
```

- Block packets from 127.0.0.1

Code changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "ipaddress"
    inputipaddr = "127.0.0.1"
    action = "block"
```

Result:

```
root@tejas-Inspiron-5520:/home/tejas/PY# python UbuntuFirewall_Sample.py
udp
127.0.0.1
(32055,)
127.0.0.1 blocked
tcp
198.252.206.25
(443,)
tcp
198.252.206.25
(443,)
udp
127.0.0.1
(40243,)
127.0.0.1 blocked
tcp
127.0.0.1
(30856,)
127.0.0.1 blocked
tcp
198.252.206.25
(443,)
^CInterrupted
unbind
close
```

- Accept packets from 31.13.77.36

Code changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "ipaddress"
    inputipaddr = "31.13.77.36"
    action = "accept"
```

Result:

```
root@tejas-Inspiron-5520:/home/tejas/PY# python UbuntuFirewall_Sample.py
tcp
31.13.77.36
(80,)
Packet accepted
tcp
31.13.77.36
(80,)
Packet accepted
```

- Block packets from 31.13.77.36

Code changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "ipaddress"
    inputipaddr = "31.13.77.36"
    action = "block"
```

Result:

```
root@tejas-Inspiron-5520:/home/tejas/PY# python UbuntuFirewall_Sample.py
tcp
31.13.77.36
(80,)
31.13.77.36 blocked
tcp
31.13.77.36
(80,)
31.13.77.36 blocked
```

c) Port Number

- Accept packets from port number 80

Code Changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "portnum"
    inputportnum = "80"
    action = "accept"
```

Result:

```
tcp
172.217.11.164
(80,)
Packet accepted
tcp
172.217.11.164
(80,)
Packet accepted
```

- Block packets from port number 80

Code Changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "portnum"
    inputportnum = "80"
    action = "block"
```

Result:

```
root@tejas-Inspiron-5520:/home/tejas/PY# python UbuntuFirewall_Sample.py
tcp
172.217.11.164
(80,)
80 blocked
tcp
172.217.11.164
(80,)
80 blocked
udp
10.0.0.210
(138,)
udp
172.16.35.1
(138,)
udp
192.168.208.1
(138,)
^CInterrupted
unbind
close
```

- Accept packets from port number 53

Code changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "ipaddress"
    inputportnum = "53"
    action = "accept"
```

Result:

```
10.0.0.210
(53,)
Packet accepted
icmp
10.0.0.210
0
Packet accepted
^CInterrupted
unbind
close
```

- Drop packets from port number 53

Code changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "ipaddress"
    inputportnum = "53"
    action = "block"
```

Result:

```
udp
10.0.0.210
(53,)
53 blocked
udp
10.0.0.148
(5353,)
^CInterrupted
unbind
close
```

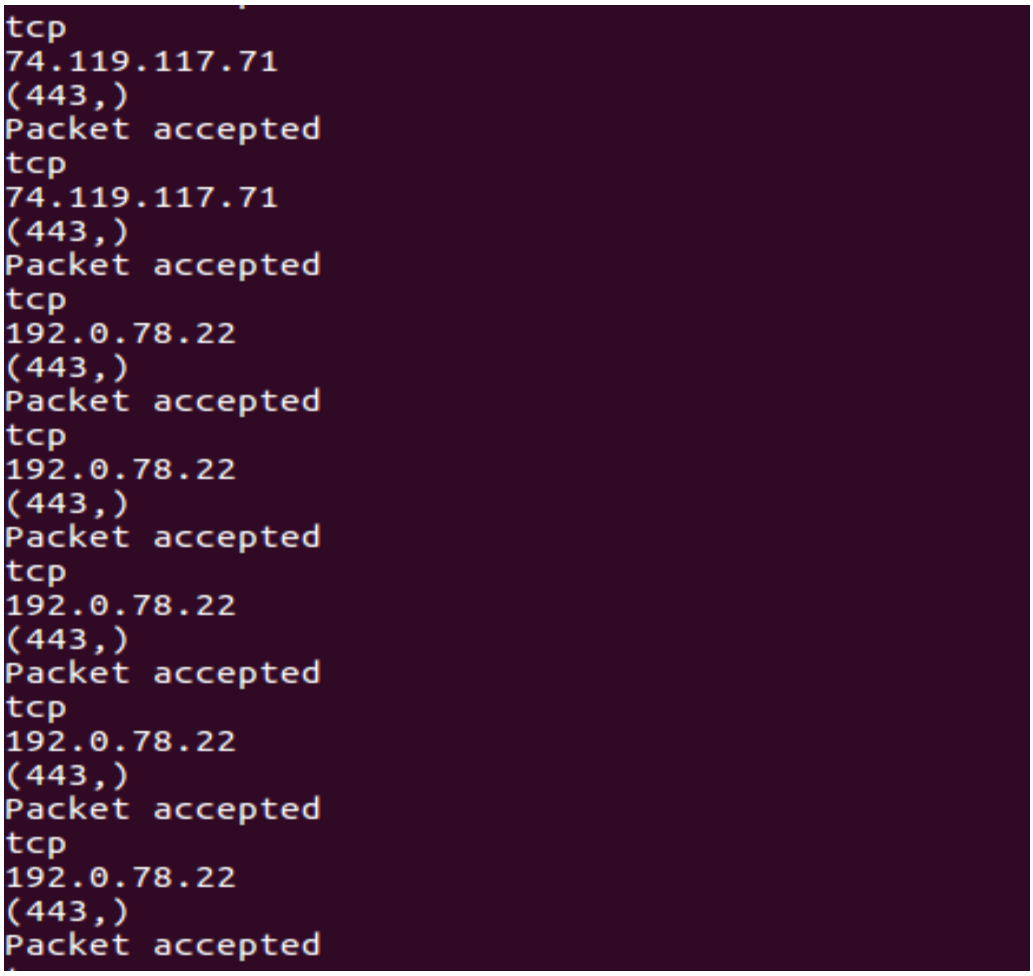
- Accept packets from port number 443

Code changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "portnum"
    inputportnum = "443"
    action = "accept"
```

Result:



- Drop packets from port number 443

Code changes:

```
def main():
    global action, inputprotocol, actiontype, inputportnum, inputipaddr

    actiontype = "ipaddress"
    inputportnum = "443"
    action = "block"
```

Result:

```
35.190.59.101
(443,)
443 blocked
tcp
192.0.72.27
(443,)
443 blocked
tcp
74.119.117.78
(443,)
443 blocked
tcp
35.190.59.101
(443,)
443 blocked
tcp
192.0.72.27
(443,)
443 blocked
tcp
192.0.73.2
(443,)
443 blocked
```

The code is made user friendly by letting the user decide actions to be performed. Below is the enhancement of the code :

User can select 1 or 2 based on the action to be performed.

```
Enter action to be performed :
1. accept
2. block
==> 1
```

User is asked to select correct option if other than 1 or 2 is entered

```
root@tejas-Inspiron-5520:/home/tejas/PY# python UbuntuFirewall_Sample_PYTHON.py
Enter action to be performed :
1. accept
2. block
==> 3
Invalid option..... Select again

Enter action to be performed :
1. accept
2. block
```

User is then asked to select action type: Either Protocol, Port number or IP address. If Protocol is selected then user is asked to type of protocol: TCP or UDP or ICMP

```
Enter action type :
1. Protocol
2. Port number
3. IP address
==> 1
Enter protocol :
1. TCP
2. UDP
3. ICMP
==> 1
```

If user enter other than 1/2/3 then user is asked to enter a valid option:

```
Enter action type :
1. Protocol
2. Port number
3. IP address
==> 1
Enter protocol :
1. TCP
2. UDP
3. ICMP
==> 4
Invalid option..... Select again

Enter protocol :
1. TCP
2. UDP
3. ICMP
==> 3
```

User is asked to enter port number if action type **Port Number** is selected

```
Enter action to be performed :
1. accept
2. block
==> 1
Enter action type :
1. Protocol
2. Port number
3. IP address
==> 2
Enter port number: 80
```

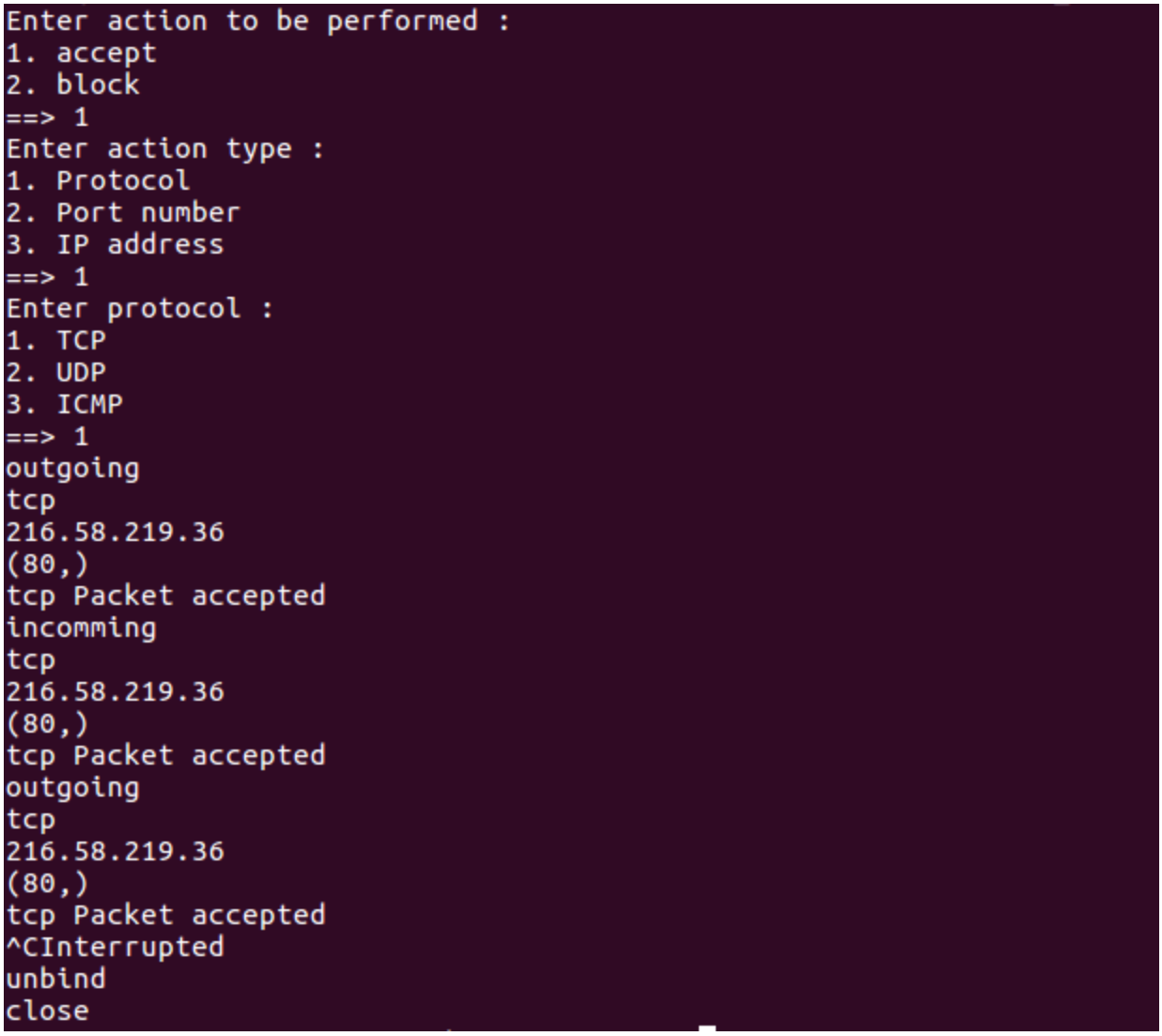
User is asked to enter IP address if action type **IP address** is selected

```
Enter action to be performed :
1. accept
2. block
==> 1
Enter action type :
1. Protocol
2. Port number
3. IP address
==> 3
Enter ip address: 127.0.0.1
```

2) Code can be enhanced by capturing and filtering outgoing traffic. Code changes are done to determine whether the packet is incoming or outgoing and further processing is done based on that.

```
#check whether the packet is incomming or outgoing
if(socket.inet_ntoa(str(pkt)[12:16])==deviceipaddr):
    print ("outgoing")
    f.handle_packet("outgoing", str(pkt))
else:
    print ("incomming")
    f.handle_packet("incoming", str(pkt))
```

Result:



A feature is added such that if rule is to accept ICMP packets then all other protocol packets will be discarded.

Code changes:

```
elif action == "accept":
    if actiontype == "protocol":
        if inputprotocol == f.protocolname:  #TCP, UDP, ICMP
            payload.set_verdict(nfqueue.NF_ACCEPT)
            print (f.protocolname + " Packet accepted")
        else:
            payload.set_verdict(nfqueue.NF_DROP)
            print (f.protocolname + " Packet blocked")

    elif actiontype == "ipaddress":

        if inputipaddr == socket.inet_ntoa(f.srcipaddress):  #IP address
            payload.set_verdict(nfqueue.NF_ACCEPT)
            print (socket.inet_ntoa(f.srcipaddress) + " accepted")
        else:
            payload.set_verdict(nfqueue.NF_DROP)
            print (socket.inet_ntoa(f.srcipaddress) + " blocked")

    elif actiontype == "portnum":
        if int(inputportnum) in f.srcportnum:
            payload.set_verdict(nfqueue.NF_ACCEPT)
            print (inputportnum + " accepted")
        else:
            payload.set_verdict(nfqueue.NF_DROP)
            print (inputportnum + " blocked")
```

Result:

```
root@tejas-Inspiron-5520:/home/tejas/PY# python UbuntuFirewall_Sample_PYTHON.py
Enter action to be performed :
1. accept
2. block
==> 1
Enter action type :
1. Protocol
2. Port number
3. IP address
==> 1
Enter protocol :
1. TCP
2. UDP
3. ICMP
==> 3
incomming
tcp
127.0.0.1
(30856,)
tcp Packet blocked
incomming
udp
127.0.0.1
(25067,)
udp Packet blocked
incomming
udp
127.0.0.1
(27594,)
udp Packet blocked
```