Simulate TCP-SYN flooding with IP Spoofing between Vms

Introduction

The purpose of this simulation is to understand how TCP-SYN flooding works and its effect.

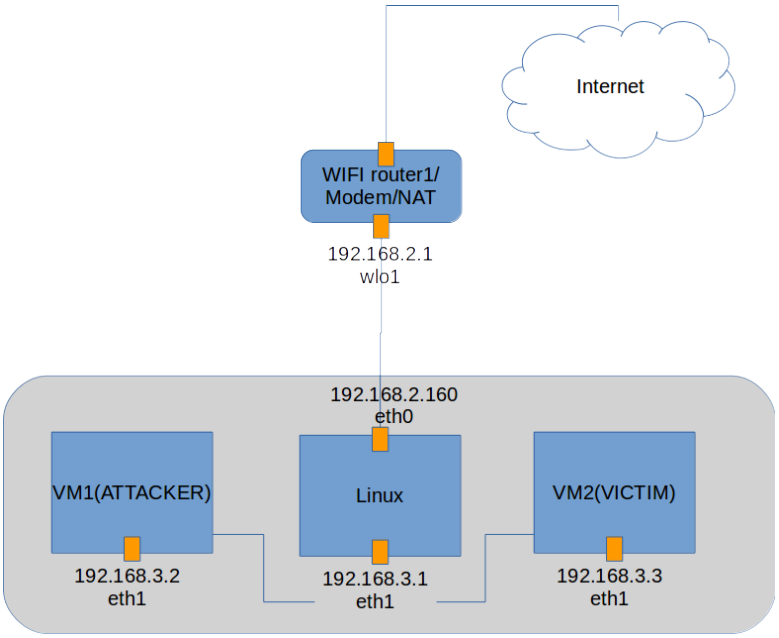
Simulation Environment

Host machine hardware info	VM Guest machine hardware info
H/W path Device Class Description ========== system Computer /0 bus Motherboard /0/0 memory 15GiB System memory /0/1 processor Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz	vagrant@victim:~$ sudo lshw -short H/W path Device Class Description ========== system VirtualBox () /0 bus VirtualBox /0/0 memory 128KiB BIOS /0/1 memory 363MiB System memory /0/2 processor Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz
Host cpu info	Guest cpu info

```
exinton@exinton-HP-ZBook-15:~/OpenStackCookbook$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 60
Model name:            Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz
Stepping:              3
CPU MHz:               3341.812
CPU max MHz:           3400.0000
CPU min MHz:           800.0000
BogoMIPS:              4789.04
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              6144K
NUMA node0 CPU(s):     0-7
```

```
vagrant@victim:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                2
On-line CPU(s) list:   0,1
Thread(s) per core:    1
Core(s) per socket:    2
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 60
Stepping:              3
CPU MHz:               2394.286
BogoMIPS:              4788.57
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              6144K
NUMA node0 CPU(s):     0,1
```

Logical network topology



Comparison between different programming

I tried Java, C and Python for the TCP flooding client.

Since Java doesn't support raw socket, an alternative way to realize tcp flooding in java is JNI.

My first draft was C, and improved to multi-threading later.

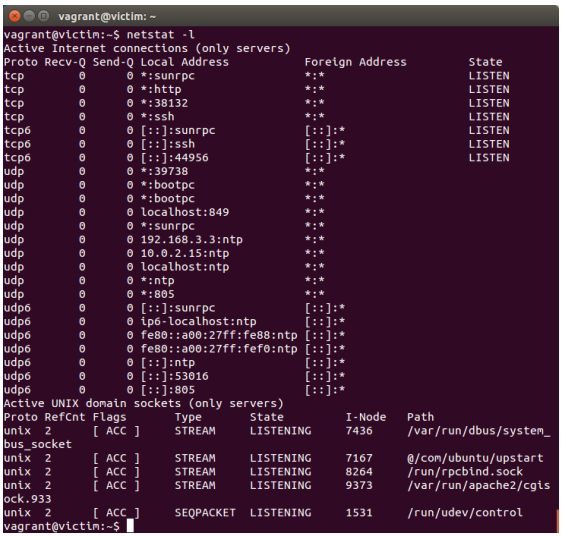
Another version is written in python, I add multi-threading as well.

In order to collect the Syn-received msg easily, I use a ruby script to fetch the data automatically.

Source code file version:

Source File name	Main function	feature	Attack effect
synflood_ipspoofing.c	Initial attack	rawsocket	Longest DoS time
Syn.c	Initial attack	Multi-threading	medium DoS time
Tcp-flooding.py	Initial attack	rawsocket	medium DoS time
Tcp-flooding-multi-threading.py	Initial attack	Multi-threading	Longest DoS time
check_syn_flood.rb	Counter the syn-recv TCP session number		

Test environment preparation:

1. Verify 80 port on virtual mahcine victim	

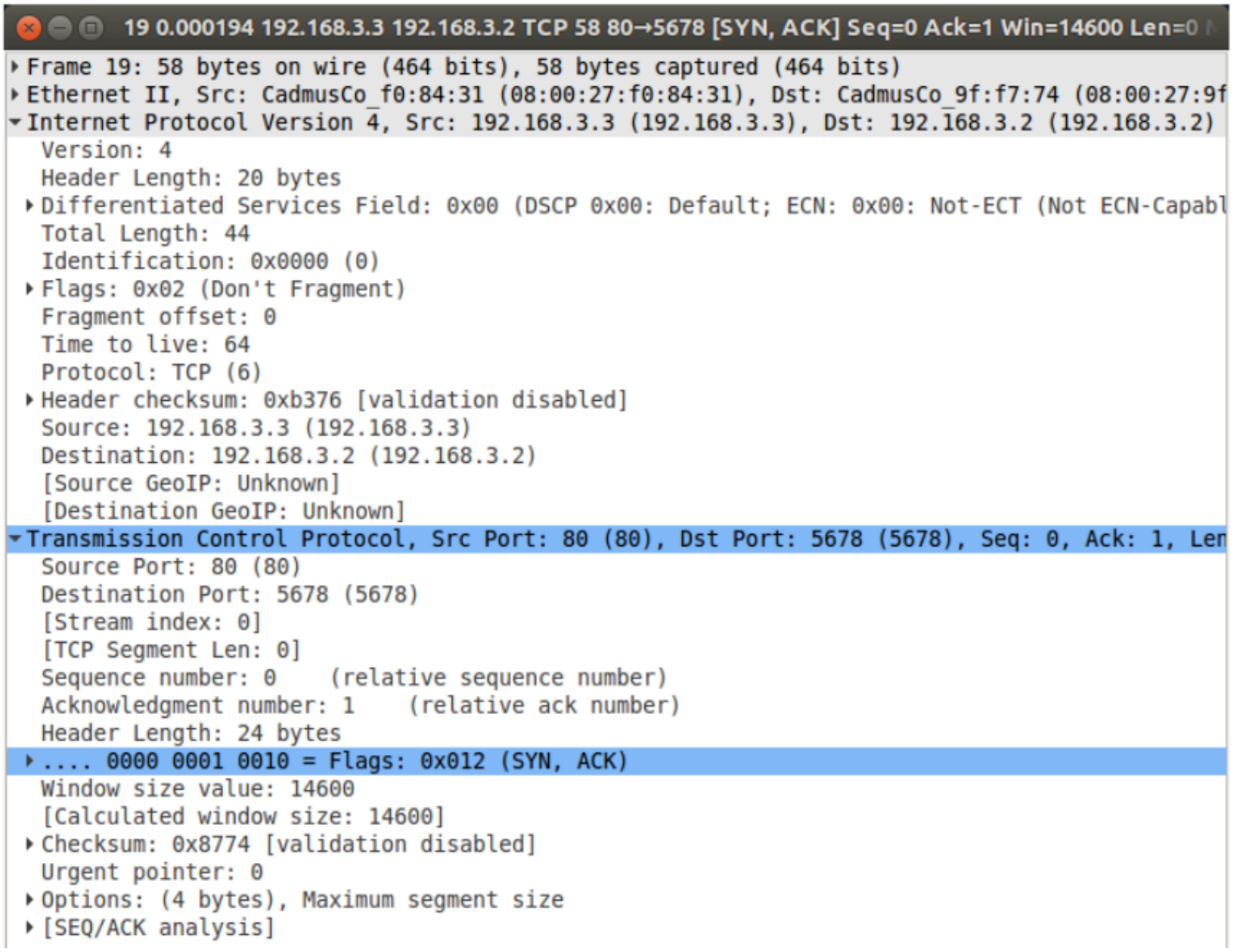
2. set up vm via vagrant	exinton@exinton-HP-ZBook-15: ~/workspace/python-tcp-flooding exinton@exinton-HP-ZBook-15:~/workspace/python-tcp-flooding$ vagrant global-status id name provider state directory -- e055d31 controller virtualbox poweroff /home/exinton/openstack e4f91ba default virtualbox poweroff /home/exinton/mininet a2f22b6 default virtualbox running /home/exinton/Project/tcpsynflood c2453ad default virtualbox running /home/exinton/Project/victim d73abb0 default virtualbox poweroff /home/exinton/Project/mysql The above shows information about all known Vagrant environments on this machine. This data is cached and may not be completely up-to-date. To interact with any of the machines, you can go to that directory and run Vagrant, or you can use the ID directly with Vagrant commands from any directory. For example: "vagrant destroy 1a2b3c4d" exinton@exinton-HP-ZBook-15:~/workspace/python-tcp-flooding$
3.open 80 port on victim vm	vagrant@victim: ~ vagrant@victim:~$ netstat -l Active Internet connections (only servers) Proto Recv-Q Send-Q Local Address Foreign Address State tcp 0 0 *:sunrpc *:* LISTEN tcp 0 0 *:http *:* LISTEN tcp 0 0 *:38132 *:* LISTEN tcp 0 0 *:ssh *:* LISTEN tcp6 0 0 [::]:ssh [::]:* LISTEN tcp6 0 0 [::]:44956 [::]:* LISTEN udp 0 0 *:39738 *:* udp 0 0 *:bootpc *:* udp 0 0 *:bootpc *:* udp 0 0 localhost:849 *:* udp 0 0 *:sunrpc *:* udp 0 0 192.168.3.3:ntp *:* udp 0 0 10.0.2.15:ntp *:* udp 0 0 localhost:ntp *:* udp 0 0 *:ntp *:* udp 0 0 *:805 *:* udp6 0 0 [::]:sunrpc [::]:* udp6 0 0 ip6-localhost:ntp [::]:* udp6 0 0 fe80::a00:27ff:fe88:ntp [::]:* udp6 0 0 fe80::a00:27ff:fef0:ntp [::]:* udp6 0 0 [::]:ntp [::]:* udp6 0 0 [::]:53016 [::]:* udp6 0 0 [::]:805 [::]:*

Verify the attack tool

In order to verify the attack works well, several tests are needed. The simplest way is to initial an syn message to victim from a real IP address, and capture the reply from the victim.

The to verify the attack tool is that sometimes the wrong checksum in TCP part may lead to victim dropping the syn message. The Checksum part in IP header doesn't matter since Linux kernel will fill it automatically.

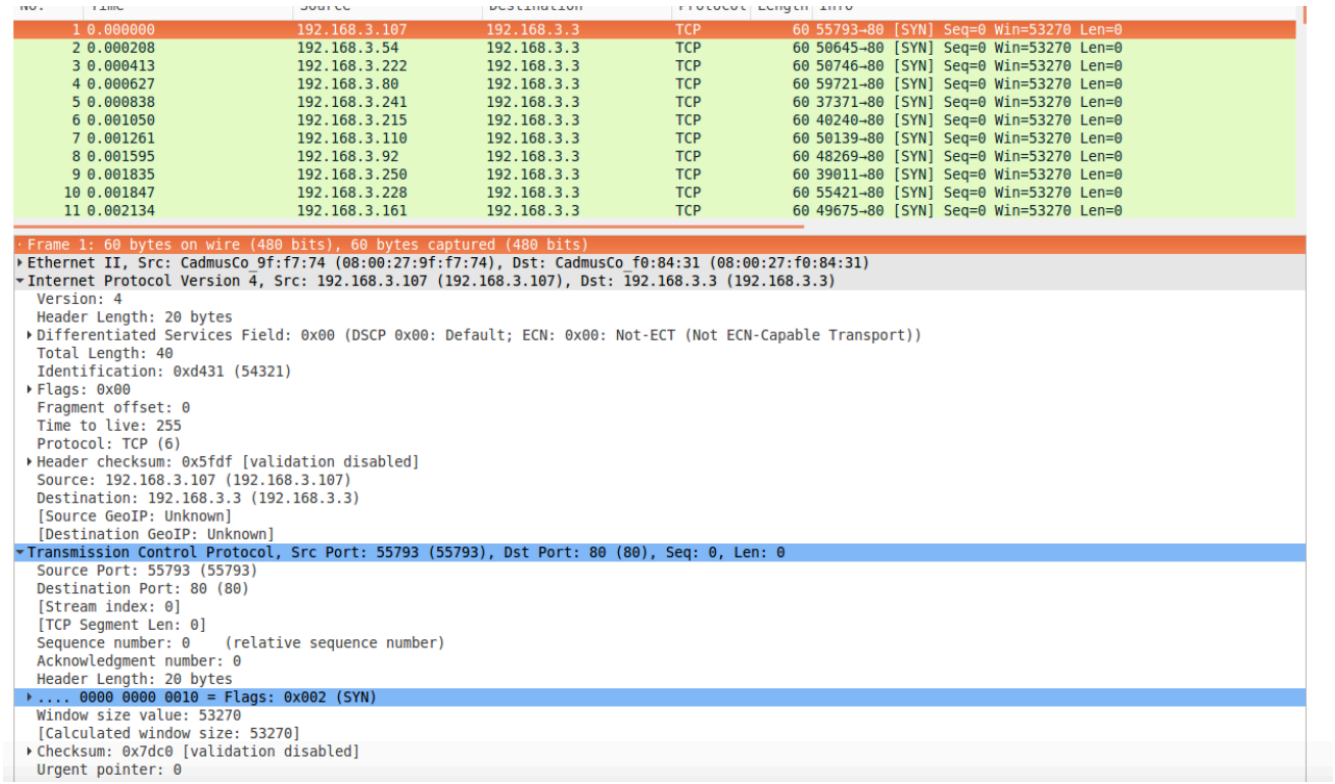
In the following picture, the victim(192.168.3.2) replies syn,ack as response, which proofs that the attack tool works perfect.



Syn attack example

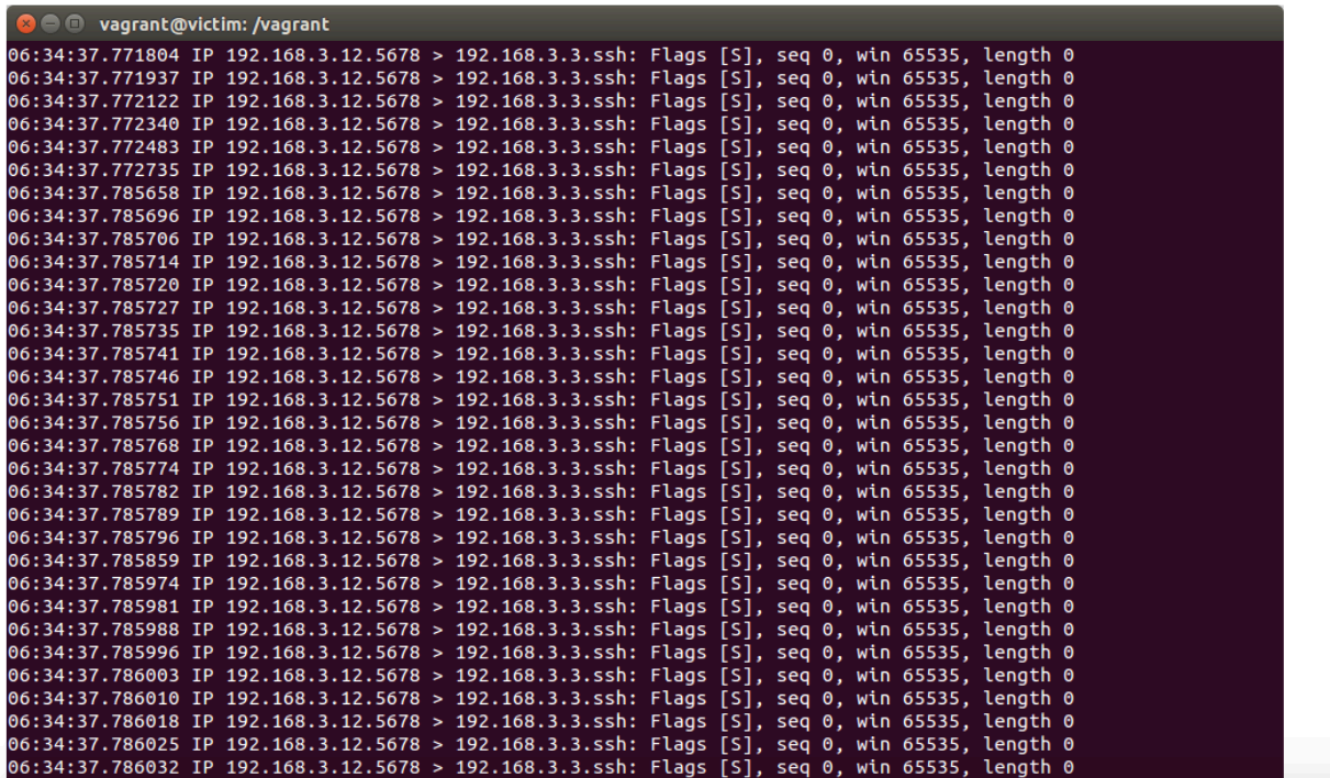
Syn attack with IP spoofing and dynamic source port

This attack dynamically changes its source port to prevent detection.
As the following diagram indicates, the source port is different at each syn attack.



Syn attack with IP spoofing and static source port

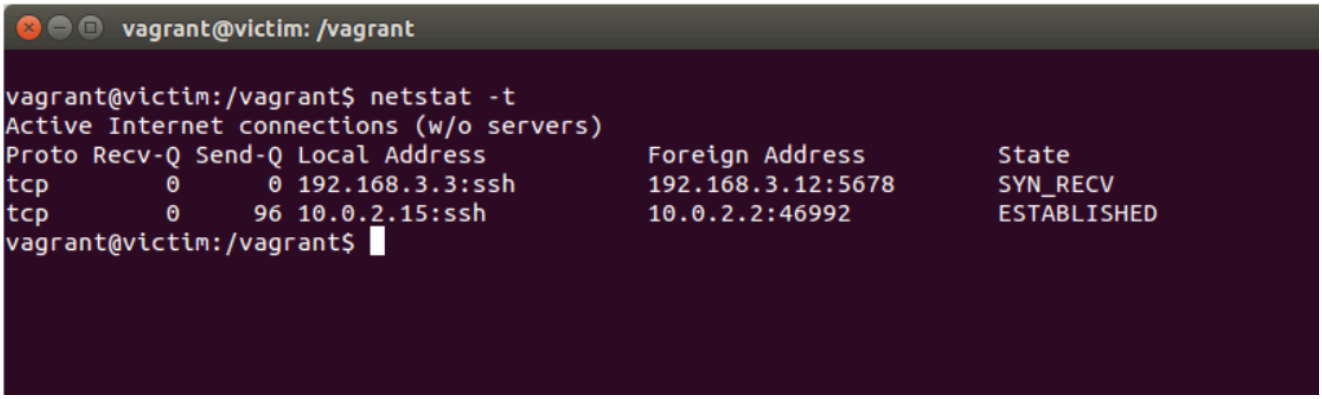
The simplest syn attack which has fixed combination of source ip address and ip ports.



verify the attack with netstat

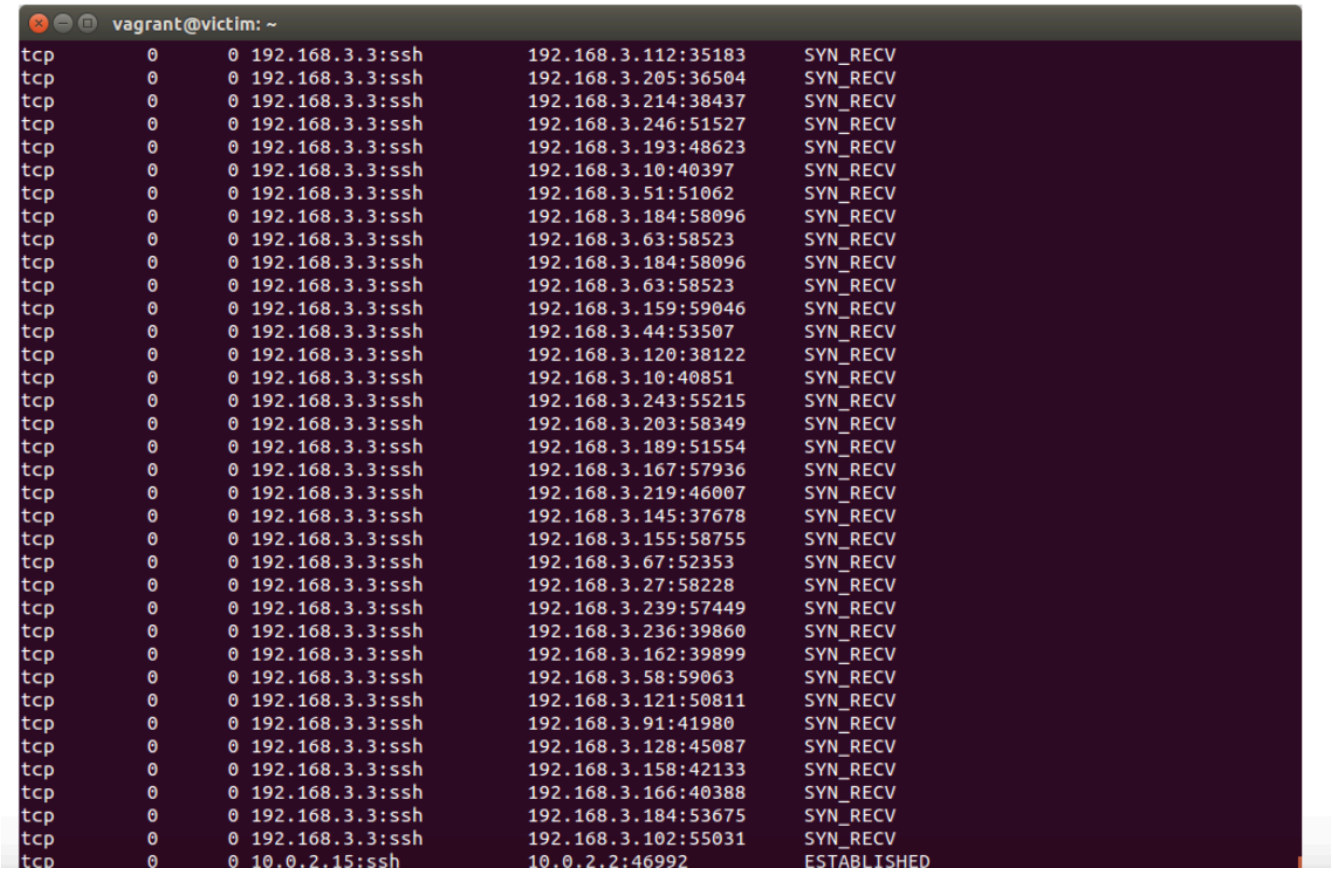
attack from static ip address

the netstat indicates that there is only one syn-recv since the ip address and ports combination is static.



attack from dynamic source ip and source port

the netstat indicates that coming ip address and ip ports changed dynamically.



quantity of tcp syn-rev

With the help of ruby , we get the average of hanging tcp session is 60 when launching dynamic source ip attack.

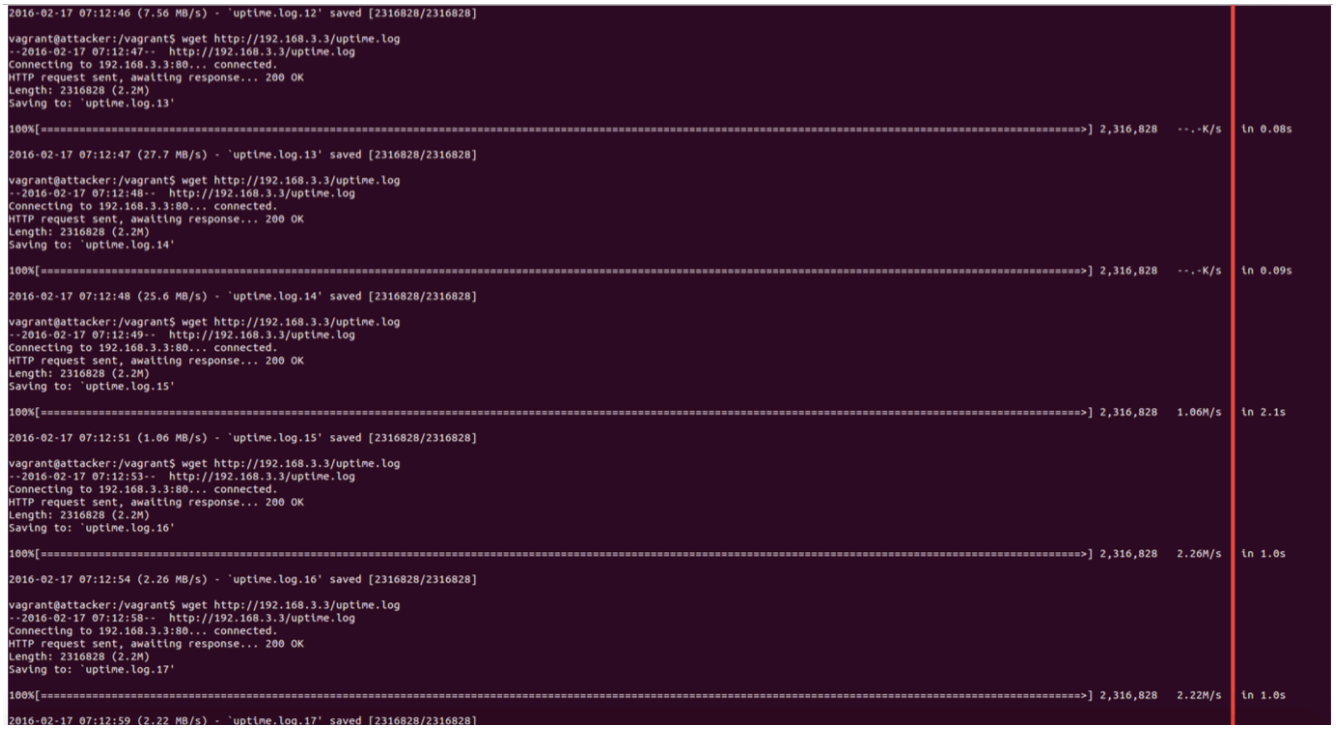
```
vagrant@victim: ~
vagrant@victim:~$ sudo ruby ./check_syn_flood.rb -w 500 -c 1000
SYN Count: 64
vagrant@victim:~$ sudo ruby ./check_syn_flood.rb -w 500 -c 1000
SYN Count: 63
vagrant@victim:~$ sudo ruby ./check_syn_flood.rb -w 500 -c 1000
SYN Count: 62
vagrant@victim:~$ sudo ruby ./check_syn_flood.rb -w 500 -c 1000
SYN Count: 60
vagrant@victim:~$ sudo ruby ./check_syn_flood.rb -w 500 -c 1000
SYN Count: 59
vagrant@victim:~$ sudo ruby ./check_syn_flood.rb -w 500 -c 1000
SYN Count: 63
vagrant@victim:~$ sudo ruby ./check_syn_flood.rb -w 500 -c 1000
SYN Count: 58
vagrant@victim:~$ sudo ruby ./check_syn_flood.rb -w 500 -c 1000
SYN Count: 65
vagrant@victim:~$ sudo ruby ./check_syn_flood.rb -w 500 -c 1000
SYN Count: 67
vagrant@victim:~$ sudo ruby ./check_syn_flood.rb -w 500 -c 1000
SYN Count: 71
vagrant@victim:~$ sudo ruby ./check_syn_flood.rb -w 500 -c 1000
SYN Count: 69
vagrant@victim:~$ sudo ruby ./check_syn_flood.rb -w 500 -c 1000
SYN Count: 72
vagrant@victim:~$ sudo ruby ./check_syn_flood.rb -w 500 -c 1000
SYN Count: 74
vagrant@victim:~$ sudo ruby ./check_syn_flood.rb -w 500 -c 1000
SYN Count: 76
vagrant@victim:~$
```

verify the attack effect

the average time of wget a file before syn flooding is about 0.04s.

```
2016-02-17 07:15:05 (32.9 MB/s) - `uptime.log.20' saved [2316828/2316828]

vagrant@attacker:/vagrant$ wget http://192.168.3.3/uptime.log
--2016-02-17 07:15:05--  http://192.168.3.3/uptime.log
Connecting to 192.168.3.3:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2316828 (2.2M)
Saving to: `uptime.log.21'

100%[======================================================================================================================================================>] 2,316,828   --.-K/s   in 0.04s

2016-02-17 07:15:06 (55.6 MB/s) - `uptime.log.21' saved [2316828/2316828]

vagrant@attacker:/vagrant$ wget http://192.168.3.3/uptime.log
--2016-02-17 07:15:06--  http://192.168.3.3/uptime.log
Connecting to 192.168.3.3:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2316828 (2.2M)
Saving to: `uptime.log.22'

100%[======================================================================================================================================================>] 2,316,828   --.-K/s   in 0.05s

2016-02-17 07:15:06 (43.2 MB/s) - `uptime.log.22' saved [2316828/2316828]

vagrant@attacker:/vagrant$ wget http://192.168.3.3/uptime.log
--2016-02-17 07:15:07--  http://192.168.3.3/uptime.log
Connecting to 192.168.3.3:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2316828 (2.2M)
Saving to: `uptime.log.23'

100%[======================================================================================================================================================>] 2,316,828   --.-K/s   in 0.04s

2016-02-17 07:15:07 (53.1 MB/s) - `uptime.log.23' saved [2316828/2316828]

vagrant@attacker:/vagrant$ wget http://192.168.3.3/uptime.log
--2016-02-17 07:15:08--  http://192.168.3.3/uptime.log
Connecting to 192.168.3.3:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2316828 (2.2M)
Saving to: `uptime.log.24'

100%[======================================================================================================================================================>] 2,316,828   --.-K/s   in 0.05s

2016-02-17 07:15:08 (40.5 MB/s) - `uptime.log.24' saved [2316828/2316828]

vagrant@attacker:/vagrant$ wget http://192.168.3.3/uptime.log
--2016-02-17 07:15:09--  http://192.168.3.3/uptime.log
Connecting to 192.168.3.3:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2316828 (2.2M)
Saving to: `uptime.log.25'
```

the average time of wget a file during syn flooding is about 1s.



the top commands during syn flooding

you can also noticed the softirq load increase a lot because it has to handle the tcp socket.

```
top - 17:10:57 up 39 min,  1 user,  load average: 0.00, 0.01, 0.05
Tasks:  94 total,   2 running,  92 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.2%sy,  0.0%ni, 86.0%id,  0.0%wa,  0.0%hi, 13.8%si,  0.0%st
Mem:    372268k total,   350244k used,    22024k free,    17080k buffers
Swap:   786428k total,      340k used,   786088k free,   201300k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    3 root      20   0     0    0    0 S    0  0.0   0:00.19 ksoftirqd/0
    7 root      20   0     0    0    0 R    0  0.0   0:00.25 rcu_sched
    1 root      20   0 24452 2180 1236 S    0  0.6   0:00.62 init
    2 root      20   0     0    0    0 S    0  0.0   0:00.00 kthreadd
    5 root       0 -20     0    0    0 S    0  0.0   0:00.00 kworker/0:0H
```

the top commands before syn flooding

```
top - 17:12:42 up 40 min,  1 user,  load average: 0.00, 0.01, 0.05
Tasks:  94 total,   1 running,  93 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni, 99.8%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:    372268k total,   350560k used,    21708k free,    17100k buffers
Swap:   786428k total,      340k used,   786088k free,   201300k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    8 root      20   0     0    0    0 S    0  0.0   0:00.47 rcuos/0
   32 root      20   0     0    0    0 S    0  0.0   0:00.92 kworker/0:1
   52 root      20   0     0    0    0 S    0  0.0   0:00.47 kworker/u4:1
    1 root      20   0 24452 2180 1236 S    0  0.6   0:00.62 init
    2 root      20   0     0    0    0 S    0  0.0   0:00.00 kthreadd
```

summary

Syn flooding is very cheap. With several lines of code, you can slow down the victim's network performance a lot.
In terms of programming, the multi-threading doesn't help to boost the syn flood effect mainly because the bottle neck is not the socket system which needs to be shared between different threads.