Packet Sniffer implementation on Python platform

Lekhani Ray 14BCE0155
Deepa V 14BCB0103
S K Janani 16BCE0618

Course Code: CSE1004
Faculty: Dr. Jaisankar N
Slot: L57+L58

Project undertaken in completion of 1 credit for Project based learning under Dr Jaisankar N, for Networks and communications

Abstract: Packet sniffer is a technique of monitoring every packet that crosses the network. By using this developers can easily obtain the information of the packet, such as structures, types, sizes and data. Consequently, developers will find and correct errors rapidly and conveniently. Packet sniffer is a program running in a network attached device that passively receives all data link layer frames passing through the device's network adapter. It is also known as network analyzer, protocol analyzer or packet analyzer, or for particular types of networks, an Ethernet sniffer or wireless sniffer. The packet sniffer captures the data that is addressed to other machines, saving it for later analysis. Most of the time, we system administrators use packet sniffing to troubleshoot network problems like finding out why traffic is so slow in one part of the network. Capturing, or sniffing, network traffic is invaluable for network administrators troubleshooting network problems, security engineers investigating network security issues, developers debugging communication protocol implementations, or anyone trying to learn how their networks work. Because attackers use sniffers for network reconnaissance and to intercept transmitted credentials and data, learning about the capabilities and limitations of packet sniffers is an important facet of understanding the security risks.

Keywords: packet sniffer,trouble shooting,Ethernet sniffer,debugging

1 Introduction

Packet sniffing is a technique of monitoring every packet that crosses the network. A packet sniffer is a piece of software or hardware that monitors all network traffic. This is unlike standard network hosts that only receive traffic sent specifically to them. The security threat presented by sniffers is their ability to capture all incoming and

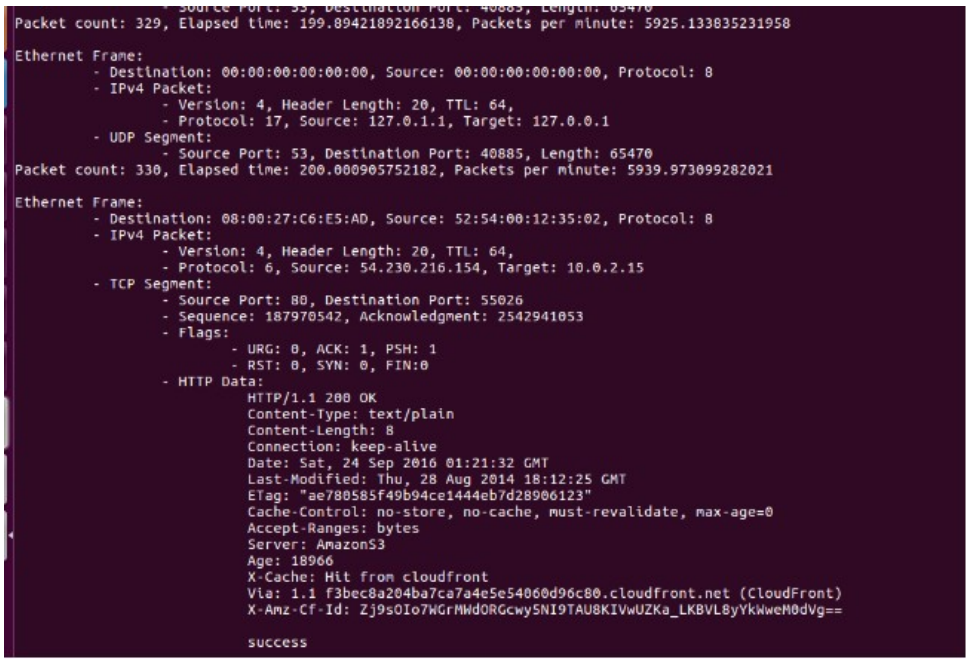
outgoing traffic, including clear-text passwords and user names or other sensitive material. In theory, it's impossible to detect these sniffing tools because they are passive in nature, meaning that they only collect data. While they can be fully passive, some aren't therefore they can be detected. This paper discusses the different packet sniffing methods and explains how Anti-Sniff tries to detect these sniffing programs.

2 Related work

Anshul Gupta of Suresh Gyan University,India has done a research study on packet sniffing tool TCPDUMP. A paper on Anti Packet Sniffing by rupal sinha and D.K Mishra of Acropolis Institue of Technology and Reasearch Indore deals with how the sniffer captures each packet and eventually decodes and analyzes its content.so, in information security that many of the features that make using computers easier or more efficient and tools used to protect and secure the network can also be used to exploit and compromise the same computers and networks.Using information captured by the packet sniffer an administrator can identify erroneous packets and use the data to pinpoint bottlenecks and help maintain efficient network data transmission.Typically the packet sniffer would only capture packets that were intended for the machine in question,however if placed in promiscuous mode, the packet sniffer is also capable of capturing all packets traversing network regardless of destination. The paper Packet Sniffer Detection with AntiSniff by Ryan Spangler, University of Wisconsin – Whitewater discusses the different methods that AntiSniff uses to detect these sniffing programs.

3 Dataset and the method

The sample output looks something like this:



Under Ethernet frame the source and destination IP is displayed. Along with the IPv4 version, header length, time to live, protocol,source and target.
The protocol is identified as either FTP,UDP,TCP or SMTP. Then further information based on the protocol is displayed. This could be the source port,destination port along with sequence,acknowledgement and flags.
After the header is read, the actual data is sniffed out and displayed

4 Working of a packet sniffer

A packet sniffer works by looking at every packet sent in the network, including packets not intended for itself. This is accomplished in a variety of ways. These sniffing methods will be described below. Sniffers also work differently depending on the type of network they are in Shared Ethernet: In a shared Ethernet environment, all hosts are connected to the same bus and compete with one another for bandwidth. In such an environment packets meant for one machine are received by all the other machines. Thus, any machine in such an environment placed in promiscuous mode will be able to capture packets meant for other machines and can therefore listen to all the traffic on the network. Switched Ethernet: An Ethernet environment in which the hosts are connected to a switch instead of a hub is called a Switched Ethernet. The switch maintains a table keeping track of each computer's MAC address and delivers packets destined for a particular machine to the port on which that machine is connected. The switch is an intelligent device that sends packets to the destined computer only and does not broadcast to all the machines on the network, as in the previous case. This switched Ethernet environment was intended for better network performance, but as an added benefit, a machine in promiscuous mode will not work here. As a result of this, most network administrators assume that sniffers don't work in a Switched Environment.

5 Sniffing methods

There are three types of sniffing methods. Some methods work in non-switched networks while others work in switched networks. The sniffing methods are: IP-based sniffing, MAC-based sniffing, and ARP-based sniffing.

5.1.1 IP-based sniffing

This is the original way of packet sniffing. It works by putting the network card into promiscuous mode and sniffing all packets matching the IP address filter. Normally, the IP address filter isn’t set so it can capture all the packets. This method only works in nonswitched networks.

5.1.2 MAC-based sniffing

This method works by putting the network card into promiscuous mode and sniffing all packets matching the MAC address filter.

5.1.3 ARP-based sniffing

This method works a little different. It doesn't put the network card into promiscuous mode. This isn't necessary because ARP packets will be sent to us. This happens because the ARP protocol is stateless. Because of this, sniffing can be done on a switched network. To perform this kind of sniffing, you first have to poison the ARP cache1 of the two hosts that you want to sniff, identifying yourself as the other host in the connection. Once the ARP caches are poisoned, the two hosts start their connection, but instead of sending the traffic directly to the other host it gets sent to us. We then log the traffic and forward it to the real intended host on the other side of the connection. This is called a man-in-the-middle attack.

7 Conclusions and future work

When computers communicate over networks, they normally just listen to the traffic specifically for them. However, network cards have the ability to enter promiscuous mode, which allows them to listen to all network traffic regardless of if it's directed to them. Packet sniffers can capture things like clear-text passwords and usernames or other sensitive material. Because of this packet sniffers are a serious matter for network security. Fortunately, not all sniffers are fully passive. Since they aren't tools like AntiSniff can detect them. Since sniffing is possible on non-switched and switched networks, it's a good practice to encrypt your data communications..

Source Code: (for reference only)

```
import socket
import struct
import textwrap
#from general import *
#from networking.ethernet import Ethernet
#from networking.ipv4 import IPv4
#from networking.icmp import ICMP
#from networking.tcp import TCP
#from networking.udp import UDP
#from networking.pcap import Pcap
#from networking.http import HTTP

TAB_1 = '\t - '
TAB_2 = '\t\t - '
TAB_3 = '\t\t\t - '
TAB_4 = '\t\t\t\t - '

DATA_TAB_1 = '\t   '
DATA_TAB_2 = '\t\t   '
DATA_TAB_3 = '\t\t\t   '
DATA_TAB_4 = '\t\t\t\t   '


def main():
    pcap = Pcap('capture.pcap')
    conn = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))

    while True:
        raw_data, addr = conn.recvfrom(65535)
        pcap.write(raw_data)
        eth = Ethernet(raw_data)

        print('\nEthernet Frame:')
        print(TAB_1 + 'Destination: {}, Source: {}, Protocol: {}'.format(eth.dest_mac, eth.src_mac,
eth.proto))

        if eth.proto == 8:
            ipv4 = IPv4(eth.data)
            print(TAB_1 + 'IPv4 Packet:')
            print(TAB_2 + 'Version: {}, Header Length: {}, TTL: {},'.format(ipv4.version,
ipv4.header_length, ipv4.ttl))
            print(TAB_2 + 'Protocol: {}, Source: {}, Target: {}'.format(ipv4.proto, ipv4.src,
ipv4.target))

                    if ipv4.proto == 1:
                icmp = ICMP(ipv4.data)
                print(TAB_1 + 'ICMP Packet:')
                print(TAB_2 + 'Type: {}, Code: {}, Checksum: {},'.format(icmp.type, icmp.code,
icmp.checksum))
```

```
            print(TAB_2 + 'ICMP Data:')
            print(format_multi_line(DATA_TAB_3, icmp.data))

                elif ipv4.proto == 6:
            tcp = TCP(ipv4.data)
            print(TAB_1 + 'TCP Segment:')
            print(TAB_2 + 'Source Port: {}, Destination Port: {}'.format(tcp.src_port, tcp.dest_port))
            print(TAB_2 + 'Sequence: {}, Acknowledgment: {}'.format(tcp.sequence,
tcp.acknowledgment))
            print(TAB_2 + 'Flags:')
            print(TAB_3 + 'URG: {}, ACK: {}, PSH: {}'.format(tcp.flag_urg, tcp.flag_ack,
tcp.flag_psh))
            print(TAB_3 + 'RST: {}, SYN: {}, FIN:{}'.format(tcp.flag_rst, tcp.flag_syn,
tcp.flag_fin))

                elif ipv4.proto == 17:
            udp = UDP(ipv4.data)
            print(TAB_1 + 'UDP Segment:')
            print(TAB_2 + 'Source Port: {}, Destination Port: {}, Length: {}'.format(udp.src_port,
udp.dest_port, udp.size))


                else:
            print(TAB_1 + 'Other IPv4 Data:')
            print(format_multi_line(DATA_TAB_2, ipv4.data))

        else:
        print('Ethernet Data:')
        print(format_multi_line(DATA_TAB_1, eth.data))




def __init__(self, raw_data):

         dest, src, prototype = struct.unpack('! 6s 6s H', raw_data[:14])

         self.dest_mac = get_mac_addr(dest)
        self.src_mac = get_mac_addr(src)
        self.proto = socket.htons(prototype)
        self.data = raw_data[14:]

def __init__(self, raw_data):
    version_header_length = raw_data[0]
    self.version = version_header_length >> 4
    self.header_length = (version_header_length & 15) * 4
    self.ttl, self.proto, src, target = struct.unpack('! 8x B B 2x 4s 4s', raw_data[:20])
    self.src = self.ipv4(src)
    self.target = self.ipv4(target)
    self.data = raw_data[self.header_length:]
```

```
def __init__(self, raw_data):
        self.type, self.code, self.checksum = struct.unpack('! B B H', raw_data[:4])
        self.data = raw_data[4:]


def __init__(self, raw_data):
        (self.src_port, self.dest_port, self.sequence, self.acknowledgment, offset_reserved_flags) =
struct.unpack(
            '! H H L L H', raw_data[:14])
        offset = (offset_reserved_flags >> 12) * 4
        self.flag_urg = (offset_reserved_flags & 32) >> 5
        self.flag_ack = (offset_reserved_flags & 16) >> 4
        self.flag_psh = (offset_reserved_flags & 8) >> 3
        self.flag_rst = (offset_reserved_flags & 4) >> 2
        self.flag_syn = (offset_reserved_flags & 2) >> 1
        self.flag_fin = offset_reserved_flags & 1
        self.data = raw_data[offset:]


def __init__(self, raw_data):
        self.src_port, self.dest_port, self.size = struct.unpack('! H H 2x H', raw_data[:8])
        self.data = raw_data[8:]



def __init__(self, filename, link_type=1):
        self.pcap_file = open(filename, 'wb')
        self.pcap_file.write(struct.pack('@ I H H i I I I', 0xa1b2c3d4, 2, 4, 0, 0, 65535, link_type))

def write(self, data):
        ts_sec, ts_usec = map(int, str(time.time()).split('.'))
        length = len(data)
        self.pcap_file.write(struct.pack('@ I I I I', ts_sec, ts_usec, length, length))
        self.pcap_file.write(data)

def close(self):
        self.pcap_file.close()



def __init__(self, raw_data):
        try:
            self.data = raw_data.decode('utf-8')
        except:
            self.data = raw_data




    # Returns properly formatted IPv4 address
```

```
def ipv4(self, addr):
    return '.'.join(map(str, addr))




        # TCP

            if len(tcp.data) > 0:

                # HTTP
                if tcp.src_port == 80 or tcp.dest_port == 80:
                    print(TAB_2 + 'HTTP Data:')
                    try:
                        http = HTTP(tcp.data)
                        http_info = str(http.data).split('\n')
                        for line in http_info:
                            print(DATA_TAB_3 + str(line))
                    except:
                        print(format_multi_line(DATA_TAB_3, tcp.data))
                else:
                    print(TAB_2 + 'TCP Data:')
                    print(format_multi_line(DATA_TAB_3, tcp.data))



pcap.close()
```

Acknowledgements

Thanks to;

- VIT University,Vellore TamilNadu

References

1. Dhar, Sumit. “SwitchSniff.” March 5, 2002. URL: http://www.linuxjournal.com/article.php?sid=5869 (May 11, 2003).
2. Ettercap. “ettercap.” URL: http://ettercap.sourceforge.net/ (May 11, 2003).
3. Graham, Robert. “Sniffing (network wiretap, sniffer) FAQ.” September 14, 2000. URL: http://www.robertgraham.com/pubs/sniffing-faq.html (May 11, 2003).
4. L0pht Heavy Industries. “AntiSniff – Technical Details.” July 19, 1999. URL: https://www.nsacom.net:1952/txt/Website_Mirrors/Hack/www.l0pht.com/antisniff/techpaper.html (May 11, 2003).
5. L0pht Heavy Industries. “AntiSniff – User Guide.” July 19, 1999. URL: https://www.nsacom.net:1952/txt/Website_Mirrors/Hack/www.l0pht.com/antisniff/userguide.html (May 11, 2003).
6. Zouridaki, Charikleia. “Packet Sniffing: The invisible threat and how to be protected.” October 11, 2001. URL: http://mason.gmu.edu/~czourida/publications/sniffers.pdf. (May 11, 2003).
7. Research paper proceeding of the 2nd National Conference; INDIACom-2008 by Rupal Sinha, D.K. Mishra
8. Implementation of IEEE 802.15.4 Packet Analyzer