MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Mini-Project Report
Computer Networks Lab

A.I.D.S. - An Intrusion Detection System

Submitted By:

Rushin Reddy R	200905066
Sanjna Mallappa	200905019

ABSTRACT

AIDS – An Intrusion Detection System is a simple project that detects SYN-flooding attacks and port scans. Most cyber-attacks start with reconnaissance which involves port scanning. When AIDS sees multiple SYN packets from the same source IP address to different ports on the same device in a short period of time, it knows there is an attacker on the network.

We will print the packet header information and the payload. After parsing a minimum of 100 packets AIDS will attempt to raise an alarm in the case of an attack. The working of the program is explained with the code.

Modern networked business environments require a high level of security to ensure safe and trusted communication of information between various organizations. An intrusion detection system acts as an adaptable safeguard technology for system security after traditional technologies fail. Cyber-attacks will only become more sophisticated, so it is important that protection technologies adapt along with their threats.

Tools Used:
C Programming Language
Nmap
TCPDump
LibPCAP
Linux (Ubuntu – Victim, Kali – Attacker)

ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to our Computer Networks professor, Mr. Manoj R for his guidance and support in the completion of our project. We are also grateful to the laboratory in-charge faculty for providing us with all the necessary facilities which were essential to our mini-projects successful implementation.

A special note of appreciation to our classmates and friends who helped us with exchanging useful ideas and tips whenever and wherever necessary.

TABLE OF CONTENTS

1. INTRODUCTION

1.1. GENERAL INTRODUCTION TO MAIN CONCEPTS

A typical **intrusion detection system** is a device or software application that monitors a network for malicious activity or policy violations. Any malicious activity or violation is typically reported or collected centrally using a security information and event management system.

Tcpdump is a command line utility that allows you to capture and analyze network traffic going through your system. It is often used to help troubleshoot network issues, as well as a security tool.
A powerful and versatile tool that includes many options and filters, tcpdump can be used in a variety of cases. Since it's a command line tool, it is ideal to run in remote servers or devices for which a GUI is not available, to collect data that can be analyzed later.
The capture filter syntax, along with some examples, is documented in the tcpdump man page under "expression."
Below are a few simple examples:

Expression	**Description**
----------	-----------
ip	Capture all IP packets.
tcp	Capture only TCP packets.
tcp port 80	Capture only TCP packets with a port equal to 80.
ip host 10.1.2.3	Capture all IP packets to or from host 10.1.2.3.

Working of Nmap

- The source host receives a TCP SYNACK segment from the target host. Since this means that an application is running with TCP port 6789 on the target post, **nmap** returns “open.”
- The source host receives a TCP RST segment from the target host. This means that the SYN segment reached the target host, but the target host is not running an application with TCP port 6789. But the attacker at least knows that the segments destined to the host at port 6789 are not blocked by any firewall on the path between source and target hosts. (Firewalls are discussed in Chapter 8.)
- The source receives nothing. This likely means that the SYN segment was blocked by an intervening firewall and never reached the target host.

1.2. HARDWARE AND SOFTWARE REQUIREMENTS

- C Programming Language
- Nmap
- TCPDump
- LibPCAP
- Linux (Ubuntu – Victim, Kali – Attacker)
- Wireless card or interface capable of sniffing
- Root privileges

2. PROBLEM DEFINITION

Modern networked business environments require a high level of security to ensure safe and trusted communication of information between various organizations. An intrusion detection system acts as an adaptable safeguard technology for system security after traditional technologies fail. Cyber-attacks will only become more sophisticated, so it is important that protection technologies adapt along with their threats.

To distinguish the activities of the network traffic that the intrusion and normal is very difficult and to need much time consuming. An analyst must review all the data that large and wide to find the sequence of intrusion on the network connection. Therefore, it needs a way that can detect network intrusion to reflect the current network traffics.

3. OBJECTIVES

To monitor inbound and outbound network traffic, continuously analyse activity for changes in patterns, and alert an administrator when it detects unusual behavior.

An IDS monitors traffic to and from all devices on a network. The system operates behind a firewall as a secondary filter for malicious packets and primarily looks for two suspicious clues:

- Signatures of known attacks.
- Deviations from regular activity.

An intrusion detection system relies on pattern correlation to identify threats.

The main goal of a typica IDS is to detect anomalies before hackers complete their objective. Once the system detects a threat, IDS informs the IT staff and provides the following info about the danger:

- The source address of the intrusion.
- Target and victim addresses.
- The type of threat.

We, here, only check for SYN flooding attacks and from particular IPs only.

4. METHODOLOGY

1. We begin by determining which interface we want to sniff on. In Ubuntu this is ens33. We can either define this device in a string, or we can ask pcap to provide us with the name of an interface that will do the job.
2. Initialize pcap. This is where we actually tell pcap what device we are sniffing on. We can, if we want to, sniff on multiple devices. We differentiate them using file handles. Just like opening a file for reading or writing, we must name our sniffing "session" so we can tell it apart from other such sessions.
3. In the event that we only want to sniff specific traffic we create a rule set, "compile" it, and apply it. This is a three phase process, all of which is closely related. The rule set is kept in a string, and is converted into a format that pcap can read (hence compiling it). The compilation is actually just done by calling a function within our program; it does not involve the use of an external application. Then we tell pcap to apply it to whichever session we wish for it to filter.
4. Finally, we tell pcap to enter its primary execution loop. In this state, pcap waits until it has received however many packets we want it to. Every time it gets a new packet in, it calls another function that we have already defined. The function that it calls will dissect the packet and print it to the user.
5. After our sniffing needs are satisfied, we check against patterns that show evidence of attacks and then close the session.

5. IMPLEMENTATION DETAILS

SETTING THE WIRELESS INTERFACE:

```
/* check for capture device name on command-line */
       if (argc == 2) {
        dev = argv[1];
       }
       else {
              /* find a capture device if not specified on command-line */
              dev = pcap_lookupdev(errbuf);
              if (dev == NULL) {
                     fprintf(stderr, "Couldn't find default device: %s\n",errbuf);
                     exit(EXIT_FAILURE);
              }
       }
```

OPENING THE INTERFACE FOR SNIFFING

```
handle = pcap_open_live(dev, SNAP_LEN, 1, 1000, errbuf);
    if (handle == NULL) {
        fprintf(stderr, "Couldn't open device %s: %s\n", dev, errbuf);
        exit(EXIT_FAILURE);
    }
```

FILTERING TRAFFIC

```c
    /* compile the filter expression */
    if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
        fprintf(stderr, "Couldn't parse filter %s: %s\n",
            filter_exp, pcap_geterr(handle));
        exit(EXIT_FAILURE);
    }

    /* apply the compiled filter */
    if (pcap_setfilter(handle, &fp) == -1) {
        fprintf(stderr, "Couldn't install filter %s: %s\n",
            filter_exp, pcap_geterr(handle));
        exit(EXIT_FAILURE);
    }
```

THE DETECTION

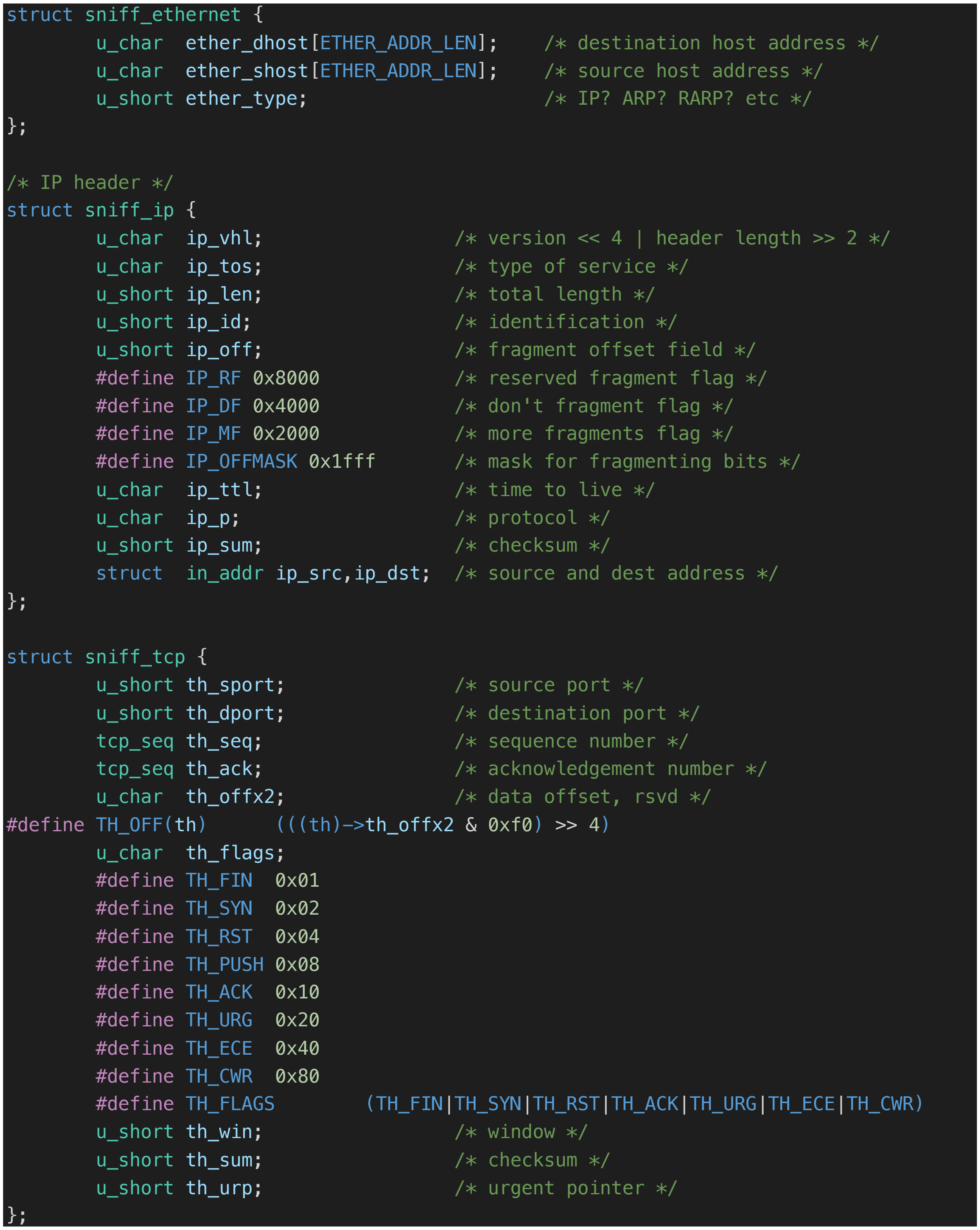
```c
/* now we can set our callback function */
    pcap_loop(handle, num_packets, got_packet, NULL);

    if(count > 40 && flag){
        printf("\n\n------------ATTACKED------------\n\n");
    }else{
        printf("\n\n------------SAFE------------\n\n");
    }


    /* cleanup */
    pcap_freecode(&fp);
    pcap_close(handle);

    printf("\nCapture complete.\n");
```

DATA STRUCTURES USED

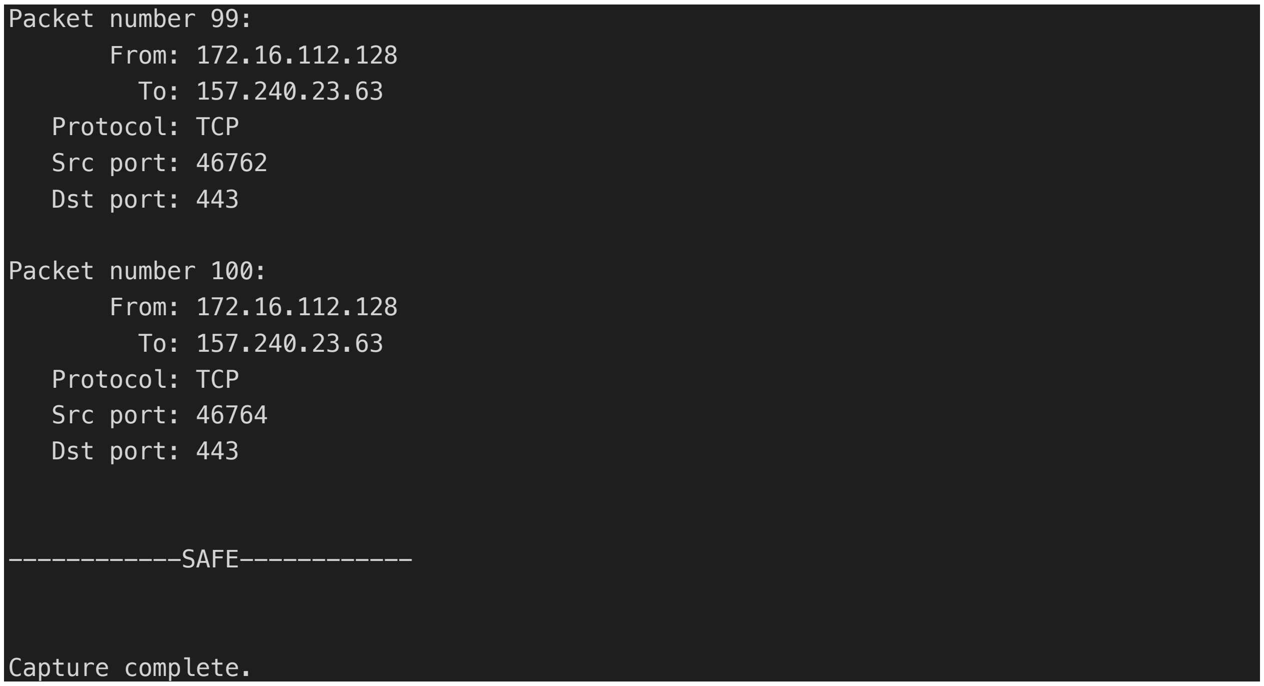


```c
struct sniff_ethernet {
        u_char  ether_dhost[ETHER_ADDR_LEN];    /* destination host address */
        u_char  ether_shost[ETHER_ADDR_LEN];    /* source host address */
        u_short ether_type;                     /* IP? ARP? RARP? etc */
};

/* IP header */
struct sniff_ip {
        u_char  ip_vhl;                 /* version << 4 | header length >> 2 */
        u_char  ip_tos;                 /* type of service */
        u_short ip_len;                 /* total length */
        u_short ip_id;                  /* identification */
        u_short ip_off;                 /* fragment offset field */
        #define IP_RF 0x8000            /* reserved fragment flag */
        #define IP_DF 0x4000            /* don't fragment flag */
        #define IP_MF 0x2000            /* more fragments flag */
        #define IP_OFFMASK 0x1fff       /* mask for fragmenting bits */
        u_char  ip_ttl;                 /* time to live */
        u_char  ip_p;                   /* protocol */
        u_short ip_sum;                 /* checksum */
        struct  in_addr ip_src,ip_dst;  /* source and dest address */
};

struct sniff_tcp {
        u_short th_sport;               /* source port */
        u_short th_dport;               /* destination port */
        tcp_seq th_seq;                 /* sequence number */
        tcp_seq th_ack;                 /* acknowledgement number */
        u_char  th_offx2;               /* data offset, rsvd */
#define TH_OFF(th)      (((th)->th_offx2 & 0xf0) >> 4)
        u_char  th_flags;
        #define TH_FIN  0x01
        #define TH_SYN  0x02
        #define TH_RST  0x04
        #define TH_PUSH 0x08
        #define TH_ACK  0x10
        #define TH_URG  0x20
        #define TH_ECE  0x40
        #define TH_CWR  0x80
        #define TH_FLAGS        (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
        u_short th_win;                 /* window */
        u_short th_sum;                 /* checksum */
        u_short th_urp;                 /* urgent pointer */
};
```

OUTPUT

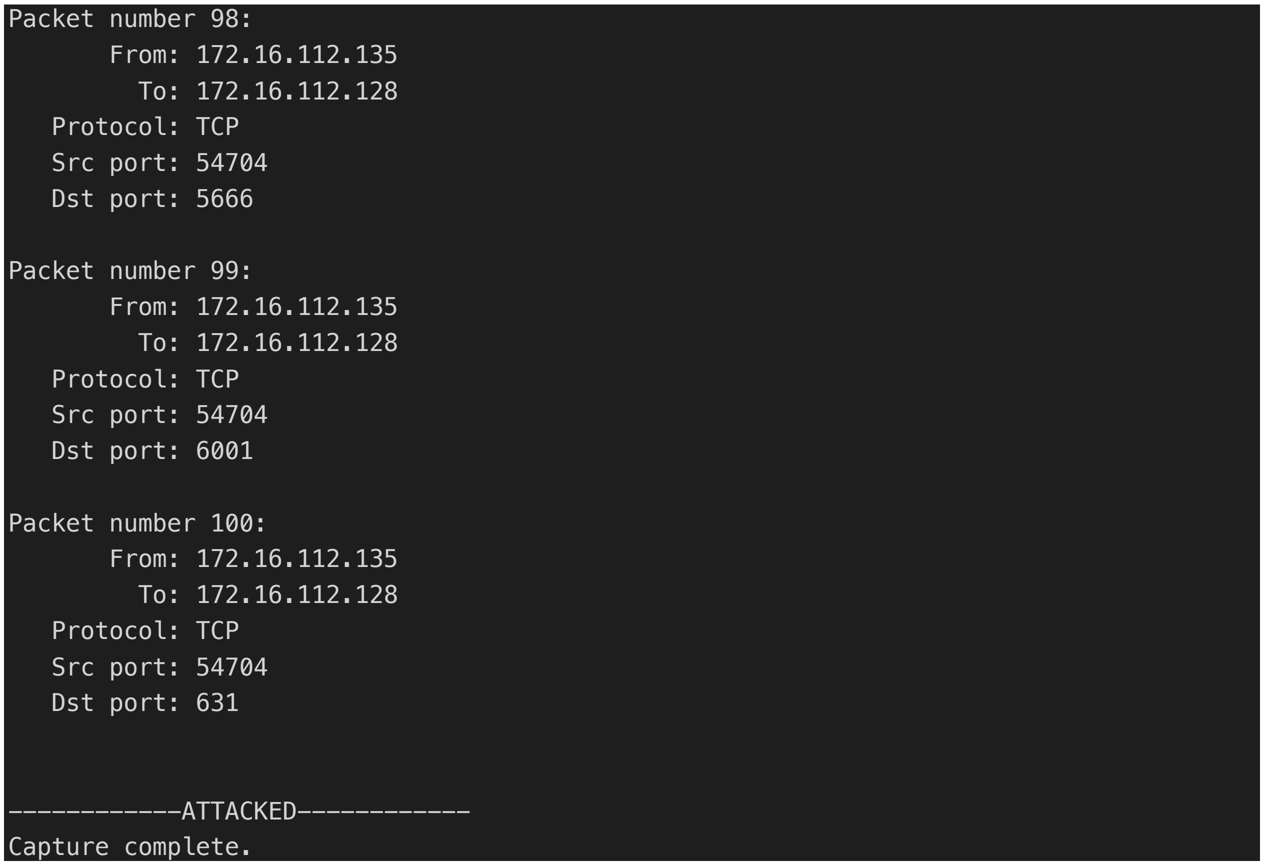


```
Packet number 99:
       From: 172.16.112.128
         To: 157.240.23.63
   Protocol: TCP
   Src port: 46762
   Dst port: 443

Packet number 100:
       From: 172.16.112.128
         To: 157.240.23.63
   Protocol: TCP
   Src port: 46764
   Dst port: 443


------------SAFE------------


Capture complete.
```



```
Packet number 98:
       From: 172.16.112.135
         To: 172.16.112.128
   Protocol: TCP
   Src port: 54704
   Dst port: 5666

Packet number 99:
       From: 172.16.112.135
         To: 172.16.112.128
   Protocol: TCP
   Src port: 54704
   Dst port: 6001

Packet number 100:
       From: 172.16.112.135
         To: 172.16.112.128
   Protocol: TCP
   Src port: 54704
   Dst port: 631


------------ATTACKED------------
Capture complete.
```

6. CONTRIBUTION SUMMARY

- **Rushin Reddy R:**
    - Preliminary research
    - Dissecting and analysis of packets
    - Applying filters and enforcing pattern match

- **Sanjna Mallappa:**
    - Preliminary research
    - Packet capture
    - Printing packet in human readable format

7. REFERENCES

- https://opensource.com/article/18/10/introduction-tcpdump
- https://www.barracuda.com/glossary/intrusion-detection-system
- https://phoenixnap.com/blog/intrusion-detection-system
- https://elf11.github.io/2017/01/22/libpcap-in-C.html
- http://tonylukasavage.com/blog/2010/12/19/offline-packet-capture-analysis-with-c-c----amp--libpcap/
- https://serverfault.com/questions/217605/how-to-capture-ack-or-syn-packets-by-tcpdump
- https://www.tcpdump.org/manpages/libpcap-1.10.1/