A  Project Report on

PACKET SNIFFER

In the partial fulfilment of the academic requirements for

M. S (Cyber Security) under Wright State University

By

Sudarshan Govindarajan               (U00793458)

Daksh Dubey (U00817740)

The following is the report that illustrates the implementation of the packet sniffer obtained by expanding the existing system.

The following files are same as by the author and source code has not been modified.

extraFunctions.c

packetcapture.h

extraFunctions.h

The following file has been modified and all the changes have been commented and been refereneced.

Packetcapture.c

Source code.



#include "packetCapture.h"

#include <string.h> //string.h has been included for the string usuage.

//Created a structure to store the source ip address of the packets

// Count variable provides the number of the failed connection in general, but it servers other purpose in few functions.

struct storage\_syn

{

char ip\_source[256];

long int count;

};

typedef struct storage\_syn node; //declaring the structure

// Declaration

node \*head;

node\* calllinkedlist();

long int k=0;

long int j=0;

// Creation of array of the declared structure

node \*syn[1000000];

#define bool int

#define true 1

#define false 0

// Created countsyn for counting of packets with failed connection.

long int countsyn=0;

char sourceipaddress[256];

char destinationipaddress[256];

//This function checks for the existing ipaddress in the stored structure if there is no existing ipaddress then new structure with new ipaddress is created.

bool syncompareipaddress(char sourceip[256])

{

char test[256];

int cmpres;

long int count;

for(k=0;k<countsyn;k++)

{

strcpy(test,syn[k]->ip\_source);

cmpres=strcmp(test,sourceip);

if(cmpres==0){

count=syn[k]->count;

count++;

syn[k]->count=count;

return true;

}

}

return false;

}

//The function stores the ipaddress for every SYN packet sent and increments the count value to show number of Transmitted SYN packets for the establishment of TCP Connection.

void createlinkedlist(char a[254])

{

bool match;

match=syncompareipaddress(a);

if(match==false){

syn[countsyn]=calllinkedlist();

strcpy(syn[countsyn]->ip\_source,a);

syn[countsyn]->count=1;

countsyn++;

}

}

//This function matches the acknowledgement packet with its corresponding SYN packet and decrements the count value because the connection has been established.

void synackcompareipaddress(char destip[256])

{

int cmpres;

long int count;

char test[256];

for(k=0;k<countsyn;k++)

{

strcpy(test,syn[k]->ip\_source);

cmpres=strcmp(test,destip);

if(cmpres==0)

{

syn[k]->count--;

}

}

}

//The function prints out the details of the IPaddresses along with the number of failed TCP Sessions intiated by the corresponding IP address.

void printfailureconnection()

{

for(k=0;k<countsyn;k++)

{

if(syn[k]->count>0)

{

printf("The failure connection for the ipaddress %s is %d \n",syn[k]->ip\_source,syn[k]->count);

}

}

}

pcap\_t\* openSource(bool inputFile, const char\* inputSource)

{

//Branch based on whether this is a file input or not.

pcap\_t\* source;

char errbuf[PCAP\_ERRBUF\_SIZE];

if (inputFile)

{

source = pcap\_open\_offline(inputSource, errbuf);

}

else

{

//Reading from a device. The true sets it into promiscuous mode.

source = pcap\_open\_live(inputSource, BUFSIZ, true, 1000, errbuf);

}

if (source == NULL)

{

printf("Issue opening %s, error is: %s", inputSource, errbuf);

exit(-1);

}

return source;

}

void installFilter(pcap\_t\* source, const char\* filter)

{

struct bpf\_program filterProgram;

//Compile and install the filter on the source.

if (pcap\_compile(source, &filterProgram, filter, 0, 0) == -1)

{

printf("Unable to compile filter, error: %s\n", pcap\_geterr(source));

printf("No filter installed.\n");

return;

}

if (pcap\_setfilter(source, &filterProgram) == -1)

{

printf("Unable to set filter, error: %s\n", pcap\_geterr(source));

printf("No filter installed.\n");

}

}

/\*

\* This method interpets the arguments given by the user and then passes

\* them to the appropriate methods.

\*/

int main ( int argc, char \*argv[] )

{

printf("Packet Analyzer\n");

//These variables track the style of input recieved.

//Only has input must be true.

bool hasInput = false;

bool hasOutput = false;

bool inputFile = false;

char\* inputSource = NULL;

char\* outputFilename = NULL;

int packetLimit = -1;

int i;

for (i = 1; i < argc; i += 2)

{

if ((i+2) > argc)

{

continue;

}

//Branch based on the flag passed.

if (strcmp(argv[i], "-i") == 0)

{

if (hasInput)

{

printf("Cannot have two inputs.\n");

exit(-1);

}

inputSource = argv[i+1];

hasInput = true;

}

else if (strcmp(argv[i], "-f") == 0)

{

if (hasInput)

{

printf("Cannot have two inputs.\n");

exit(-1);

}

inputSource = argv[i+1];

hasInput = true;

inputFile = true;

}

else if (strcmp(argv[i], "-o") == 0)

{

if (hasOutput)

{

printf("Cannot have two outputs.\n");

exit(-1);

}

outputFilename = argv[i+1];

hasOutput = true;

}

else if (strcmp(argv[i], "-l") == 0)

{

if (packetLimit > 0)

{

printf("Cannot set packet limit twice.\n");

exit(-1);

}

enum StrToIntError error = strToInt(&packetLimit, argv[i+1], 10);

if (error != SUCCESS)

{

printf("Error converting number %s\n", argv[i+1]);

exit(-1);

}

else if (packetLimit <= 0)

{

printf("Invalid packet limit (must be positive) %i\n",

packetLimit);

exit(-1);

}

}

}//End of argument for loop.

//Open the output file (if one)

FILE\* outputFile = NULL;

if (hasOutput)

{

outputFile = fopen(outputFilename, "w");

if (outputFile == NULL)

{

printf("Unable to write to output file %s\n", outputFilename);

exit(-1);

}

}

//Test with output.

if (!hasInput)

{

printf("Using default input device.\n");

char errbuf[PCAP\_ERRBUF\_SIZE];

inputSource = pcap\_lookupdev(errbuf);

if (inputSource == NULL)

{

printf("Unable to find default device, error %s\n", errbuf);

exit(-1);

}

}

if (inputFile)

{

printf("Reading from input file: %s\n", inputSource);

}

else

{

printf("Reading from interface: %s\n", inputSource);

}

if (hasOutput)

{

printf("Saving to file: %s\n", outputFilename);

}

if (packetLimit > 0)

{

printf("Packet limit: %i\n", packetLimit);

}

//Open the device.

pcap\_t\* source = openSource(inputFile, inputSource);

//Install a filter to remove all non TCP or UDP packets.

installFilter(source, "tcp or udp");

//Finally, read the packets.

readPackets(source, outputFile, packetLimit);

//Finish.

pcap\_close(source);

if (outputFile != NULL)

{

fclose(outputFile);

}

printfailureconnection();

printf("SUCCESS! Exiting...\n");

exit(0);

}

void readPackets(pcap\_t\* source, FILE\* outputFile, int packetLimit)

{

int linktype;

// Determine the datalink layer type.

if ((linktype = pcap\_datalink(source)) < 0)

{

printf("Error getting datalink size.: %s\n", pcap\_geterr(source));

return;

}

// Set the datalink layer header size.

switch (linktype)

{

case DLT\_NULL:

linkHeaderSize = 4;

break;

case DLT\_EN10MB:

linkHeaderSize = 14;

break;

case DLT\_SLIP:

case DLT\_PPP:

linkHeaderSize = 24;

break;

default:

printf("Unsupported datalink (%d)\n", linktype);

return;

}

//Loop until we run out of packets, hit the limit (if one)

//(or there's an error.)

bool isLimit = packetLimit > 0;

int currentPacket = 0;

while ((!isLimit) || (currentPacket < packetLimit))

{

//Pointers for the packet.

struct pcap\_pkthdr \*packetHeader;

const u\_char \*packetData;

//Read packet, check if error.

int result = pcap\_next\_ex(source, &packetHeader, &packetData);

if (result == 0)

{

//Timeout, loop again until we get another packet.

continue;

}

else if (result == -1)

{

printf("Error reading packet: %s\n", pcap\_geterr(source));

exit(-1);

}

else if (result == -2)

{

//Out of packets in file.

break;

}

//Handle the packet.

handlePacket(currentPacket, packetHeader, packetData, outputFile);

//Every hundred packets, flush the contents of the file to the disk.

if (outputFile != NULL && currentPacket % 100 == 0)

{

fflush(outputFile);

}

//Increment packet count.

currentPacket++;

}

}

/\*

\* Outputs a string to both stdin and the passed file.

\* string - pointer to a string to print. Must be valid.

\* file - file to write to. Must be valid.

\*/

void outputString(const char\* string, FILE\* file)

{

printf(string);

if (file != NULL)

{

fprintf(file, string);

}

}

/\*

\* This takes the packet passed by another method and if it is a TCP packet,

\* analyzes it and prints the results to the screen and an output file (if one).

\* packetHeader - packet header returned from pcap\_next\_ex function.

\* packetData - packet data returned from pcap\_next\_ex function.

\* outputFile - file to output results from (NULL if not desired)

\* linkHeaderSize - how large the header from the data link layer on this source

\* is.

\*/

void handlePacket(int packetNum, struct pcap\_pkthdr \*packetHeader, const u\_char \*packetData,

FILE\* outputFile)

{

struct ip\* ipHeader;

struct tcphdr\* tcpHeader;

struct udphdr\* udpHeader;

char sourceIP[256], destIP[256];

char\* message;

u\_char \*payload;

double timestamp = ((double)packetHeader->ts.tv\_usec);

int usec = pow(10,6);

timestamp /= usec;

timestamp += packetHeader->ts.tv\_sec;

u\_char \*packetStart = ((u\_char\*)packetData) + linkHeaderSize;

ipHeader = (struct ip\*) packetStart;

strcpy(sourceIP, inet\_ntoa(ipHeader->ip\_src));

strcpy(destIP, inet\_ntoa(ipHeader->ip\_dst));

packetStart += 4\*ipHeader->ip\_hl;

switch (ipHeader->ip\_p)

{

//Handle TCP packets.

case IPPROTO\_TCP:

tcpHeader = (struct tcphdr\*) packetStart;

//HANDLE TCP PACKETS HERE.

//Printing of Source and Destination Ip address, Sequence Number

//The following piece of code creates the structure with ipaddress and count for the SYN packet.

if( tcpHeader->syn== 1 && tcpHeader->ack==0)

{

createlinkedlist(sourceIP);

}

//The following piece of code calls the function synackcompareipaddress

if((( tcpHeader->syn)== 1) && ((tcpHeader->ack)==1))

{

synackcompareipaddress(destIP);

}

// outputString("TCP Packet.\n", outputFile);

break;

//Handle UDP packets.

case IPPROTO\_UDP:

udpHeader = (struct udphdr\*)packetStart;

//HANDLE UDP PACKETS HERE.

//outputString("TCP Packet.\n", outputFile);

break;

}

/\*

\*

\*/

}

node\* calllinkedlist()

{

node \*p,\*head;

int k=0;

if(k==0)

{

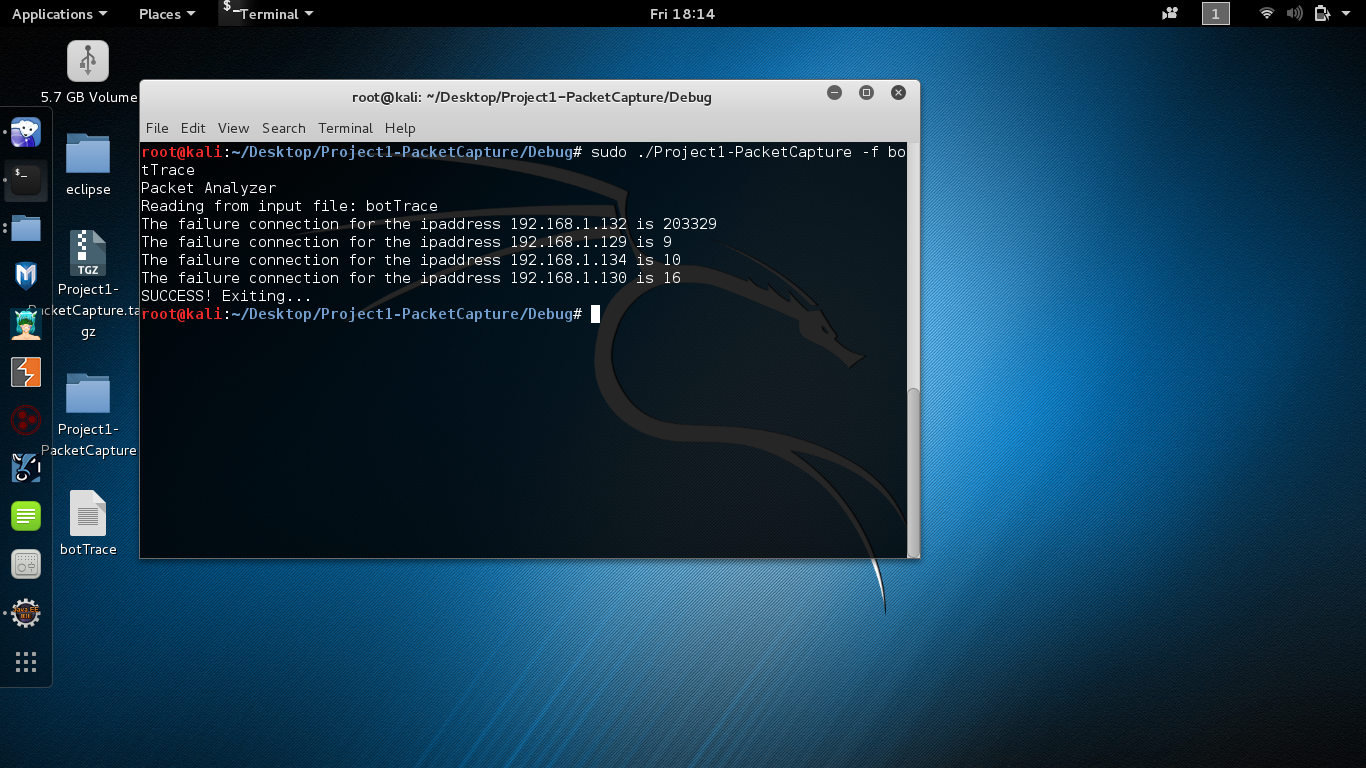
head=(node \*)malloc(sizeof(node));

p=head;

}

return p;}

Output:



**Execution**

1. Install the eclipse and download the botTrace file
2. Copy the botTrace file and save it in the Debug folder of the existing project
3. Compile the program
4. Run the program by using the code “sudo ./Project1-PacketCapture –f botTrace.

The above output will be obtained.