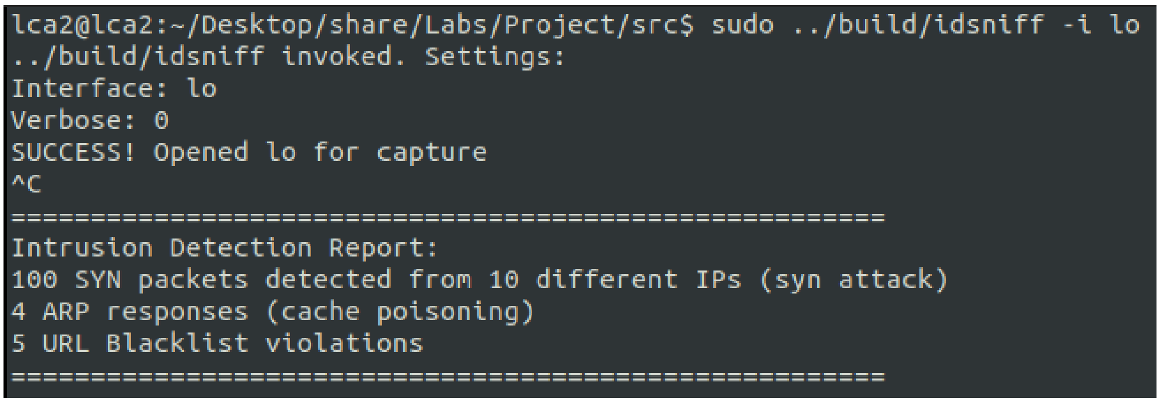
CS241 UNIVERSITY OF WARWICK

Operating Systems and Computer Networks

Joe Moore, u1917702



```
lca2@lca2:~/Desktop/share/Labs/Project/src$ sudo ../build/idsniff -i lo
../build/idsniff invoked. Settings:
Interface: lo
Verbose: 0
SUCCESS! Opened lo for capture
^C
=======================================================
Intrusion Detection Report:
100 SYN packets detected from 10 different IPs (syn attack)
4 ARP responses (cache poisoning)
5 URL Blacklist violations
=======================================================
```

A packet sniffer using the threadpool model to increase efficiency.
Includes functionality to detect SYN attacks, ARP poisoning and blacklisted IPs.

Department of Computer Science
University of Warwick
United Kingdom
November 2020

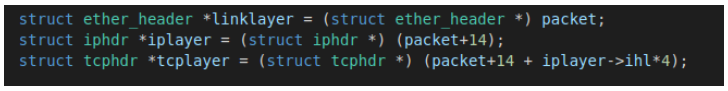
I. Introduction

This project is a packet sniffer designed in C. It uses 2 threads, applying the threadpool model, in order to achieve greater efficiency. It detects SYN attacks, ARP poisoning and blackisted URLS.

II. Design & Implementation

A. Packet breakdown

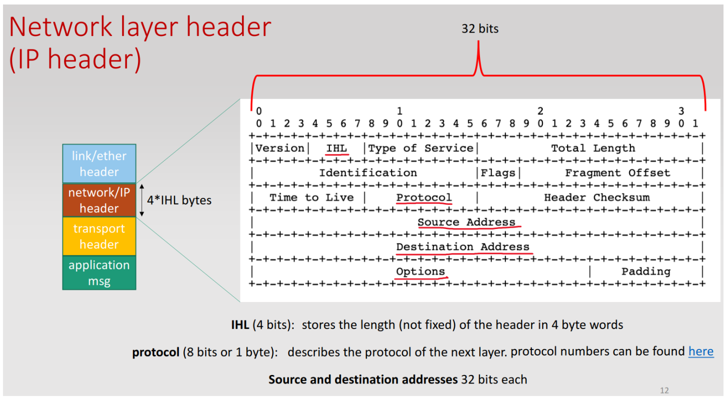
The packets are split into a `char *`, the address of the packet and a `pcap_pkthdr`, the header. These are handled within the `analyse` function. This is achieved through casting them to the structs of the different TCP/IP layers[1].



```
struct ether_header *linklayer = (struct ether_header *) packet;
struct iphdr *iplayer = (struct iphdr *) (packet+14);
struct tcphdr *tcplayer = (struct tcphdr *) (packet+14 + iplayer->ihl*4);
```

Fig. 1. How the layers of the OSI model are cast in my code

The length of link layer is 14 bytes and so we adjust he packet point by that, however the length of the ip header is dependent on the value stored in `ihl`. Hence why the packet pointer is increased in by this value multiplied 4.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IHL (4 bits): stores the length (not fixed) of the header in 4 byte words

protocol (8 bits or 1 byte): describes the protocol of the next layer. protocol numbers can be found here

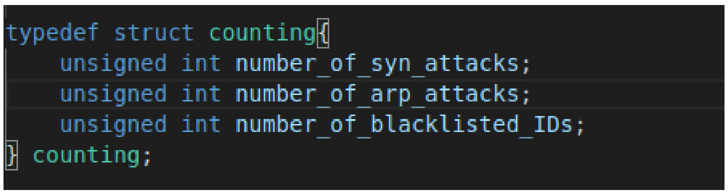
Source and destination addresses 32 bits each

Fig. 2. The layout of the network layer header, demonstrating the fact that the `ihl` byte determines the length of the network header [2]

Hence by casting these structs to the correct position of the packet we now have the packet broken down into useful structs. This is performed on each packet in `analyse()`.

B. Intrusion detection

Once we have broken down the packets we can then analyse each one for the three types of malicious activity. I have created a struct with the three counters as attributes (ints). The IPs of syn attacks are stored in a link list and counted at the end:

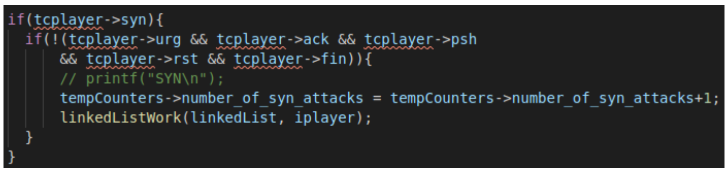


```
typedef struct counting{
    unsigned int number_of_syn_attacks;
    unsigned int number_of_arp_attacks;
    unsigned int number_of_blacklisted_IDs;
} counting;
```

Fig. 3. counting struct used to store the numbers of intrusion attacks

SYN attacks can be achieved by flooding the server with TCP SYN packets to completely overwhelm the server. SYN packets are characterised by their flags, (syn as 1 and others as 0). As such by checking the SYN bit of the TCP layer is 1 and all other flags are 0 we can easily check if a packet is a SYN attack. Therefore if a SYN attack, the packet's ip is added to a linked list and the packets counting structs syn count is incremented.[3]

ARP poisoning occurs when Address Resolution



```
if(tcplayer->syn){
  if(!(tcplayer->urg && tcplayer->ack && tcplayer->psh
      && tcplayer->rst && tcplayer->fin)){
      // printf("SYN\n");
      tempCounters->number_of_syn_attacks = tempCounters->number_of_syn_attacks+1;
      linkedListWork(linkedList, iplayer);
  }
}
```

Fig. 4. The process of detecting syn packets

Protocol (ARP) messages are sent to a network, to cause the attacker's MAC address to become associated with the IP address of another host to direct traffic to the attacker. To test for this the sniffer determines if the operation of the packet's header is 'reply' and if so will increment the counters ARP attribute.[4]

In order to prevent any attacks from suspicious domains considered to be suspicious, (according to the coursework that is `www.Google.com`. The sniffer makes use of the strstr method to find the string of this web address in the HTTP header in the Data offset byte. The destination of the TCP layer is also checked to be 80. If both conditions are met the blacklisted URL counter is incremented.

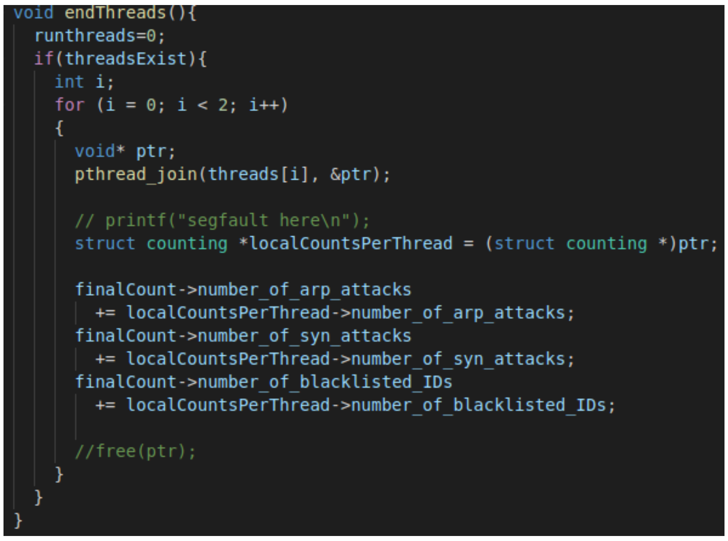
C. Threading

Threading was implemented into the design to allow two threads to concurrently assess packets and document attacks. As such I designed a linkedList struct similar to the one designed to store the the ip addresses of SYN packets, but instead to hold entire packets. Consequently `pcap_loop()` captures a packets and uses `malloc` and `memmove` to assign memory and prevent multiple pointers pointing to the same packets. Before adding it to the queue of packets by checking if the head is null if not repeating to find the last packet and setting it as that packets next value.

Upon the program starting an array of threads (from the package `pthread`) of length 2 is created. Only 2 threads are used since we are using the threadpool model. It is known that two threads per core is typically the most[5]. This is far more preferable over the one thread per x model. The reason for such a preference is that thread creation and destruction is restricted to the initial creation of the threads and final destruction of them.[6] Whereas, the creation and destruction of a new thread alongside all its associated resources can be a computationally taxing process, preventing optimum performance.[5] The VM only has one core so 2 threads were chosen.[7]

The threads upon their generation are run `threadfunction()`, this function performs the following actions:

1) Creates a new counting struct designed to store the numbers of each of the attacks that struct processes.
2) Then enters a while loop which only ends when the value of `runthreads` is set to 0. This while loop acts as follows:
    a) Lock and take the head of the queue.
    b) Call dispatch on the packet and add its results to the counting struct of the thread.
3) After the threads are closed they use `pthread_exit` to cast the counting struct to a void and return it



```
void endThreads(){
  runthreads=0;
  if(threadsExist){
    int i;
    for (i = 0; i < 2; i++)
    {
      void* ptr;
      pthread_join(threads[i], &ptr);

      // printf("segfault here\n");
      struct counting *localCountsPerThread = (struct counting *)ptr;

      finalCount->number_of_arp_attacks
        += localCountsPerThread->number_of_arp_attacks;
      finalCount->number_of_syn_attacks
        += localCountsPerThread->number_of_syn_attacks;
      finalCount->number_of_blacklisted_IDs
        += localCountsPerThread->number_of_blacklisted_IDs;

      //free(ptr);
    }
  }
}
```

Fig. 5. The combination of threads at the end

These are then both cast back and combined to give a total number of attacks in the `endthreads()` function. Since each element of the list is unique. This is then all outputted. Hence by using threading plenty of the process are able to executed in tandem saving valuable time and making the lengthiest process the adding of the packets to the queue.

The use of a `counting` variable within each thread and the merging of both of these after the process is completed is used to reduce mutex locks required and as such boost the performance.

D. Signals

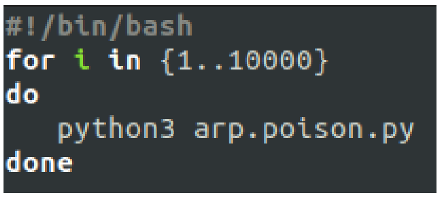
The process knows when to stop through the use of SIGNIT signals[1]. Having set Ctrl+C to be the signal to watch out for and combine threads on. [8]

III. Testing

The coursework came with a number of way of testing it manually however these were often limited and as such I enhanced them to provide more stressful tests for the system to ensure it was fully working.
Syn tests could easily be modified to include far more packets and as such be more thorough. This was done by changing the argument of the `hping` command to a higher value as follows:

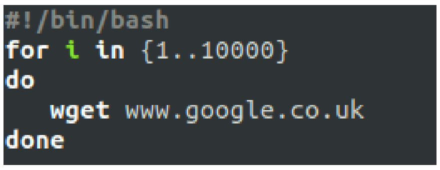
```
hping3 -c 10000 -d 120 -S -w 64 -p
80 -i u100 --rand-source localhost
```

For ARP poisoning and blacklsited IPS I had to use a custom bash function I made to test the data.



```
#!/bin/bash
for i in {1..10000}
do
   python3 arp.poison.py
done
```

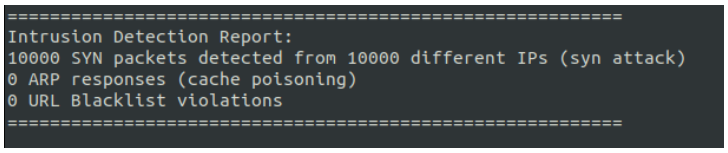
Fig. 6. Bash function to run 1000 arp attacks



```
#!/bin/bash
for i in {1..10000}
do
   wget www.google.co.uk
done
```

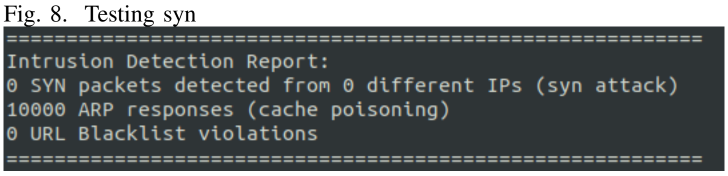
Fig. 7. Bash function to run 1000 blacklisted ips

These performed well and the desired outcomes were achieved: The use of manual testing was employed also by printing the value of parts of the packet and



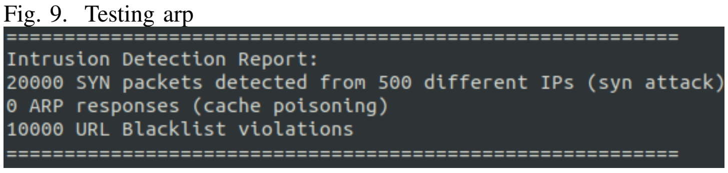
```
=========================================================
Intrusion Detection Report:
10000 SYN packets detected from 10000 different IPs (syn attack)
0 ARP responses (cache poisoning)
0 URL Blacklist violations
=========================================================
```

Fig. 8. Testing syn



```
==========================================================
Intrusion Detection Report:
0 SYN packets detected from 0 different IPs (syn attack)
10000 ARP responses (cache poisoning)
0 URL Blacklist violations
==========================================================
```

Fig. 9. Testing arp



```
=========================================================
Intrusion Detection Report:
20000 SYN packets detected from 500 different IPs (syn attack)
0 ARP responses (cache poisoning)
10000 URL Blacklist violations
=========================================================
```

Fig. 10. Testing blacklisted ips

using the `-v` extension to enable the printing and comparing them. I also tested with no threads vs 2 threads to ensure that 2 threads was actually faster.

IV. Conclusion

The use of threading is implemented such that the performance is improved and this is done successfully. The minimising of the use of mutex locks is effectively used to enhance this improvement, as is the choice of threadpool. The testing while not fully comprehensive showed that the system worked and that all the attacks were detected. Going forward these could be improved to make the checks more specific, for example doing more to evaluate whether an ARP activity is indeed malicious. However the overall success of the project in undeniable even if there exists scope for its expansion.

References

[1] B. Kernighan and D. Ritchie, *C Programming Language: C PROGRAMMING LANG _p2*. Pearson Education, 1988.

[2] A. Mukhopadhyay, "Selected topics in networking `https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs241/cn2020/`."

[3] C. M. U. S. E. Institute, "Cert advisory ca-1996-21 tcp syn flooding and ip spoofing attacks," p. 122, 12 2000.

[4] V. CRamachandran and S. Nandi, *Detecting ARP Spoofing: An Active Technique*. 12 2005, ISBN 978-3-540-30706-8.

[5] S. Nazeer, F. Bahadur, A. Khan, A. Hakeem, M. Gul, and A. Umar, "Prediction and frequency based dynamic thread pool system a hybrid model," vol. 14, p. 299, 06 2017.

[6] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs," *IEEE Transactions on Computers C-28*, vol. 9, pp. 690–691, September 1979.

[7] A. Mukhopadhyay, "Cs241 coursework `https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs241/coursework20-21//`."

[8] M. McIlroy, *A Research UNIX Reader: Annotated Excerpts from the Programmer's Manual, 1971-1986*. Computing science technical report, AT&T Bell Laboratories, 1987.