James Madison University

Integrated Science & Technology (ISAT)

ISAT 460 Networking & Cyber-Security II

Implementation of TACACS+ and Vulnerability Assessment with GNS3

Semester Project Lab Report

Author(s):
Troy Gamboa
Isaac Sumner

Submitted to:
Dr. Emil Salib

April 23, 2017

Honor Pledge:I have neither given nor received help on his lab that violates the spirit of the JMU Honor Code.

Troy Gamboa

Signature

Date

Isaac Sumner

Signature

Date

Contents

List of Figures

List of Tables

1 Introduction

When there are many network devices (routers, firewalls, switches) in an environment, it is useful to have a centralized management service for access to these devices. These services are known as Authentication, Authorization, and Accounting (AAA) services.

With TACACS+, these services are possible. A server running TACACS+ has the ability to set specific privileges commands to particular users and user groups. The TACACS+ server then logs all aspects of the AAA communication (who is logged in, what commands they are issuing, etc.).

In this lab we make use of multiple VMs (Windows 7, Ubuntu Server / Desktop, and Kali Linux) running on either an Ubuntu Desktop or OSX host. In order to test the functionality of TACACS+ in a "real-world" network situation, we also make use of the GNS3 Software. We were able to connect the virtual machines in GNS3 with the switches it provides, as well as utilizing a Cisco 3600 Router image as the TACACS+ enabled device to connect to. This router image was provided to us by Dr. Salib.

After we discovered how to implement the TACACS+ protocol in a "real-world" scenario, we began to assess the vulnerabilities that were inherent in the protocol. Some of these vulnerabilities included errors in the source code that could allow for a Denial-of-Service attack, and errors in the encryption method that TACACS+ uses for its packets that allows for a Man-in-the-Middle attack. All relevant attachments to this lab report can be found in this section: 4.2

2 Lab Network Diagrams & Tables

2.1 Exercise 1 - Network Diagrams & Tables

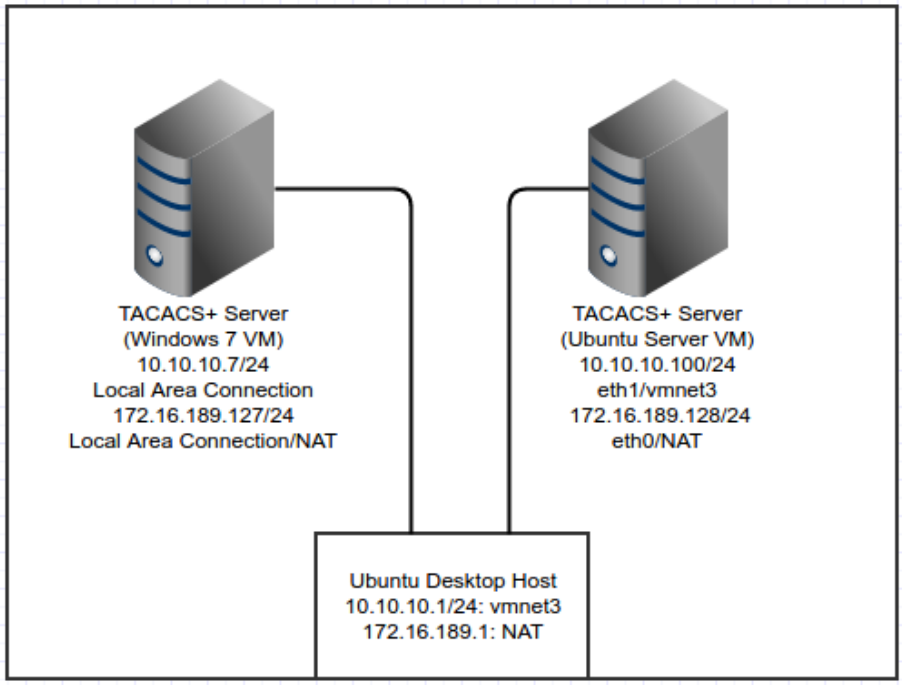


Figure 1: Ex1 Network Topology

Device	Address / mask	network	Interface
Windows 7 VM	10.10.10.7/24	vmnet3	Local Area Connection 1
Ubuntu Server VM	10.10.10.100/24	vmnet3	eth1
Ubuntu Desktop Host	10.10.10.1/24	vmnet3	vmnet3
Windows 7 VM	172.16.189.127/24	NAT	Local Area
Ubuntu Server VM	172.16.189.128/24	NAT	vmnet8

Table 1: Ex1 Network Table

2.2 Exercise 2 - Network Diagrams & Tables

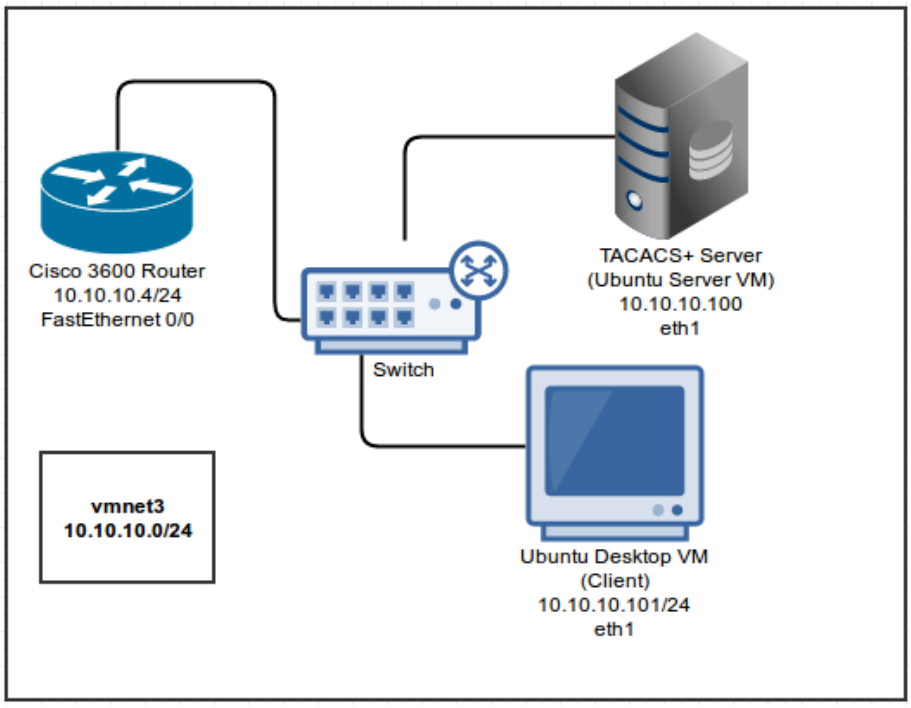


Figure 2: Ex2 Network Topology

Device	Address / mask	network	Interface
Ubuntu Desktop	10.10.10.101	vmnet3	eth1
TACACS+ Server	10.10.10.100	vmnet3	eth1
Cisco 3600 Router	10.10.10.2	vmnet3	FastEthernet0/0

Table 2: Ex2 Network Table

2.3 Exercise 3 - Network Diagrams & Tables

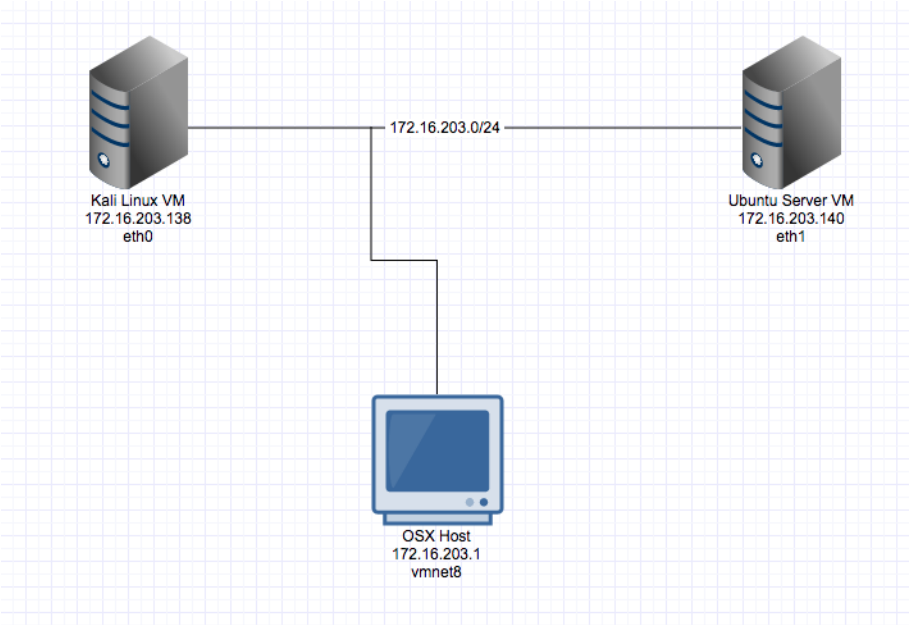


Figure 3: Ex3 Network Topology

Device	Address / mask	network	Interface
Kali Linux VM	172.16.203.138/24	NAT	eth0
Ubuntu Server VM	172.16.203.140/24	NAT	eth1
OSX Host	172.16.203.1/24	NAT	vmnet8

Table 3: Ex3 Network Table

2.4 Exercise 4 - Network Diagrams & Tables

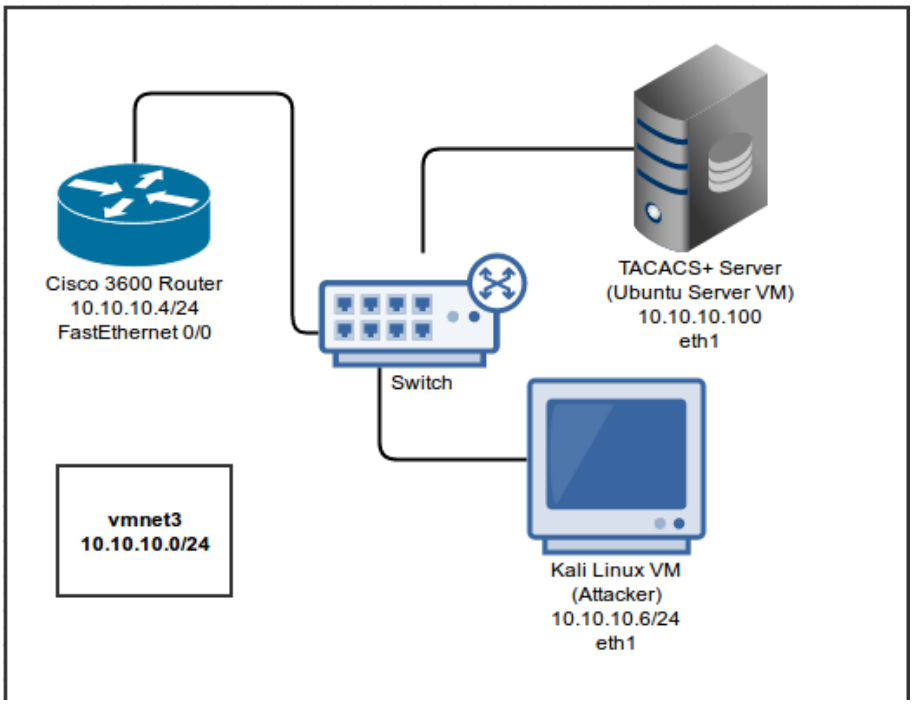


Figure 4: Ex4 Network Topology

Device	Address / mask	network	Interface
Kali Linux	10.10.10.6	vmnet3	eth1
TACACS+ Server	10.10.10.100	vmnet3	eth1
Cisco 3600 Router	10.10.10.2	vmnet3	FastEthernet0/0

Table 4: Ex4 Network Table

3 Lab Exercises : Results & Analysis

3.1 Exercise 1: Setup of TACACS+, Creating Test Users, and Configuration on Windows and Linux (Isaac)

3.1.1 Analysis & Evidence

3.1.2 Step 0: Preparation

In this lab, we were responsible for maintaining a theoretical IT department so that users could have remote access to specific network devices in a network. We found that AAA (Authentication, Authorization, and Accounting) protocols would be the best to implement in this scenario. Two protocols, RADIUS and TACACS+ began to stand out. In this lab, we chose to further investigate the services provided by TACACS+.

Q1: What does TACACS stand for, and what is it used for?
A1: TACACS stands for Terminal Access Controller Access Control System. It is used for implementing Network Access Control, and in particular, it is noted for methods of Administrative Authentication, Authorization and Accounting.

Q2: What is the difference between TACACS+ and TACACS?
A2:TACACS+(a cisco modified protocol) is the modified version of TACACS (an old open protocol). However, TACACS+ is not proprietary, as one would think. It is currently an open source protocol that can be used in a server, provided that the user has the correct environment.

Q2.5: What are the advantages that TACACS+ has over RADIUS, and what are the main purposes for using each?
A2.5: The main advantage that TACACS+ has over the RADIUS protocol is that TACACS+ separates all layers of AAA (Authentication, Authorization, and Accounting) during packet exchange. TACACS+ also has the ability to encrypt the entire packet, making it more secure than RADIUS, which only runs a hash on the password. With TACACS+ you can actually see the commands that the user has inputted to the configured device. RADIUS is most used for subscriber AAA, and TACACS+ is used for Administrative AAA. This makes TACACS more flexible and more designed to the maintaining of specific network devices. You can control who has access to those devices based on setting a permission level or setting specfic commands that that user can issue.

3.1.3 Step 1: Setup of TACACS+ Server on Windows 7 VM

We began this exercise by running the executable file provided on a Windows 7 VM.

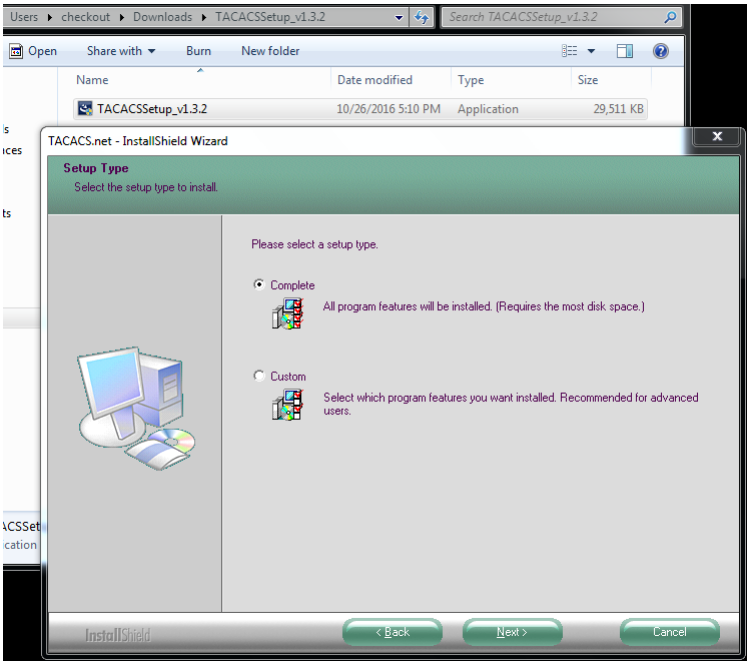


Figure 5: Shows the installation wizard of TACACS+ server on Windows 7 vm

Then, we set the secret encryption key, as stated in the lab instructions.

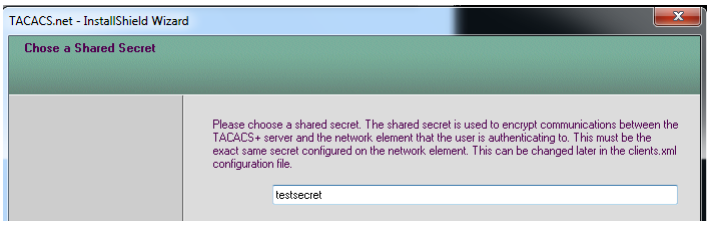


Figure 6: Setting of the secret encryption key "testsecret" in the wizard.

Then we finished the installation.

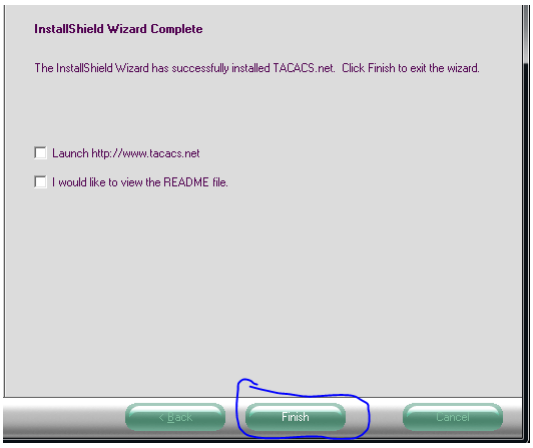


Figure 7: Finishing the installation.

Q3: Provide 2 forms of proof that the TACACS+ server is up and running on the windows 7 vm.
A3: As seen in the figures 8 and 9, We used the task manager and services window to verify that the TACACS+ server was running. Another way we could have done this is with netstat.

To check that the tacacs+ server was up and running, we navigated to "services" under the windows control panel.

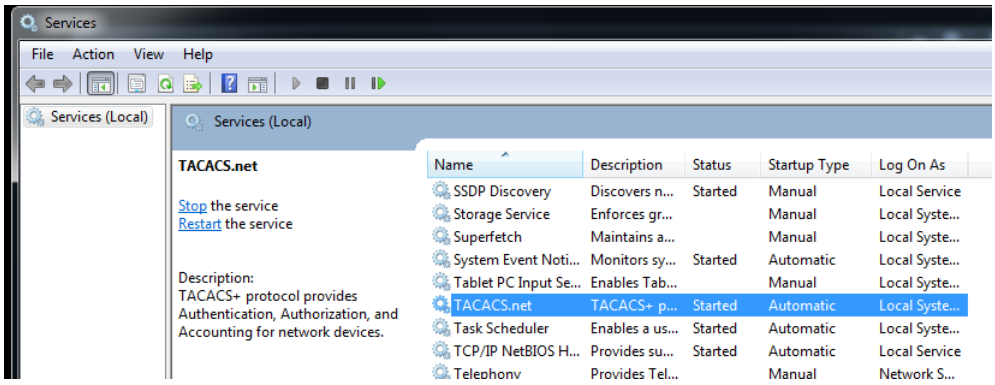


Figure 8: Checking that the server is up and running through the control panel.

We also checked if the server was running under the windows task manager.

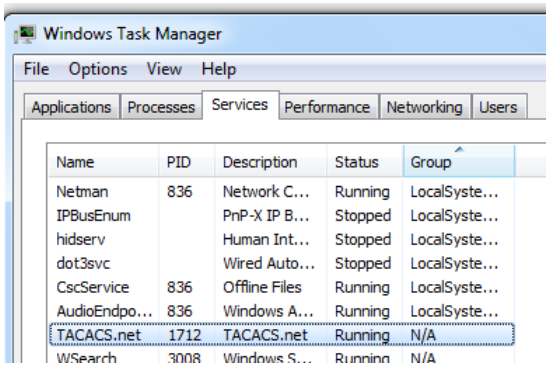


Figure 9: Checking the task manager to see that the TACACS.net server is running.

After we checked that the service was running, we navigated to the configuration files as seen in figure 10

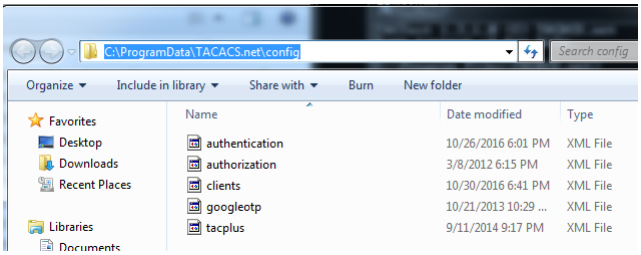


Figure 10: Shows the different configuartion files available to us in TACACS.net

Then, in the authentication.xml file, we uncommented the comments as seen in the figure 11,

```
40
41      <UserGroup>
42        <Name>Network Engineering</Name>
43        <AuthenticationType>File</AuthenticationType>
44  <!--
45      <Users>
46        <User>
47          <Name>testuser1</Name>
48          <LoginPassword ClearText="testpassword" DES=""> </LoginPassword>
49          <EnablePassword ClearText="" DES=""></EnablePassword>
50          <CHAPPassword ClearText="" DES=""> </CHAPPassword>
51          <OutboundPassword ClearText="" DES=""> </OutboundPassword>
52        </User>
53        <User>
54          <Name>user2</Name>
55          <LoginPassword ClearText="somepassword" DES=""> </LoginPassword>
56          <EnablePassword ClearText="" DES=""></EnablePassword>
57          <CHAPPassword ClearText="" DES=""> </CHAPPassword>
58          <OutboundPassword ClearText="" DES=""> </OutboundPassword>
59        </User>
60        </Users>
61  -->
62      </UserGroup>
```

Figure 11

Q4: What port does the tacacs.net tacacs+ server run on by default?
A4:tacacs+ uses port 49.

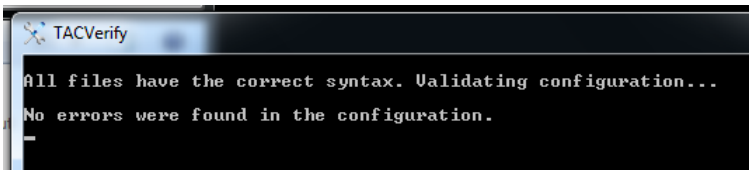


Figure 12: Shows the running of tacverify, to see if the default configurations were in working order.

Q8: How are we able to execute the `tactest` on the user "checkout" if we did not manually set the user up?
A8: By default, tacacs+ has code that allows local users located on the server to log in and use the services provided by tacacs+. When navigating to the end of the authentication file, we see that TACACS.net has automatically populated the field with the local user. See figure 13

Q9: Did the `tactest`s work? Provide screenshots for evidence.
A9: Yes, see figure 14. In the figure, you can see the successful login of the user we created, "testuser1", with the password of "testpassword".

Figure 13: Shows the part of TACACS.net that automatically populated a user "checkout", our local account credentials.

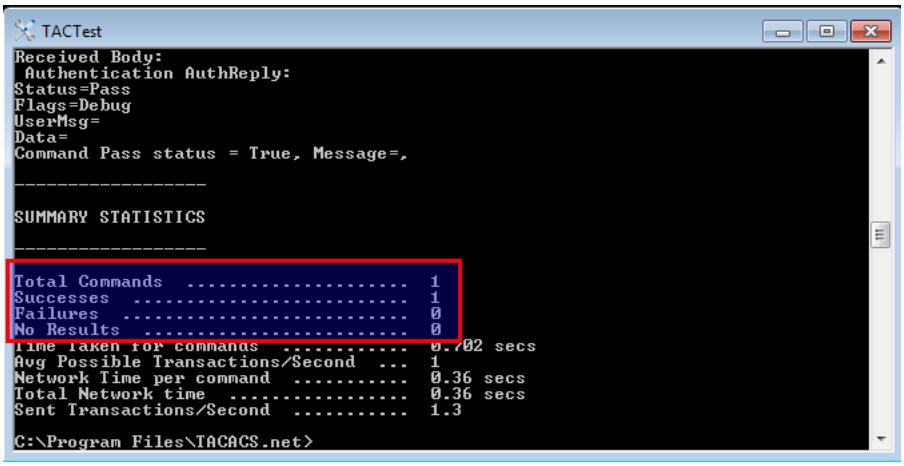


Figure 14: Proof that the tactest tool built into the TACACS.net software succeeded with the user we created.

3.1.4 Step 2: Preparation of setting up Tacacs+ server on Ubuntu Server VM

3.1.4.1 Step 1:

For this exercise we need a single Ubuntu Server vm called VM1. On this machine we run `sudo su` to become the root user.

```
checkout@ubuntu:~$ sudo su
root@ubuntu:/home/checkout#
```

Figure 15: Elevation to root

Next, we see if the tac_plus dependencies are installed by running `dpkg -s gcc bison flex`.

Figure 16: dpkg output

We see that bison and flex is not installed so we use `apt-get install` to add those packages.

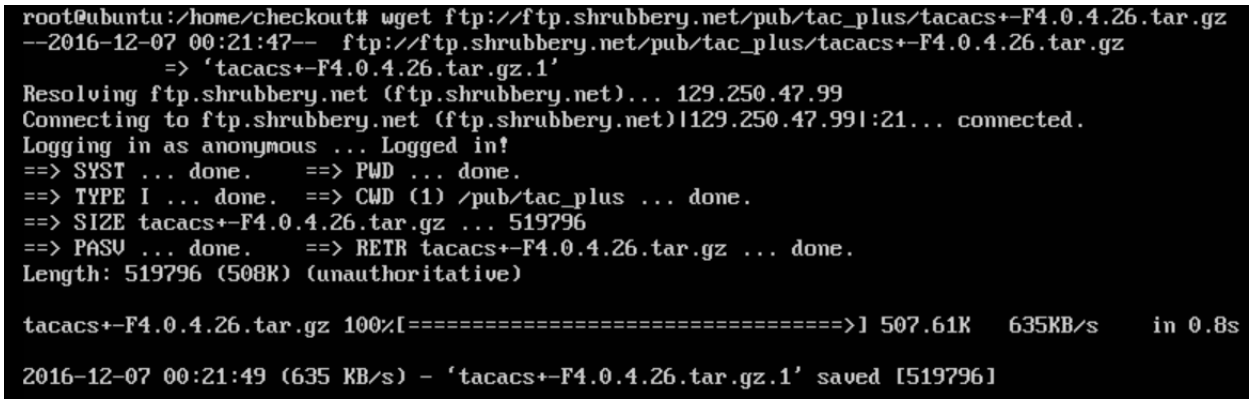


Figure 17: Downloading tacacs

Then we download and untar TACACS+.

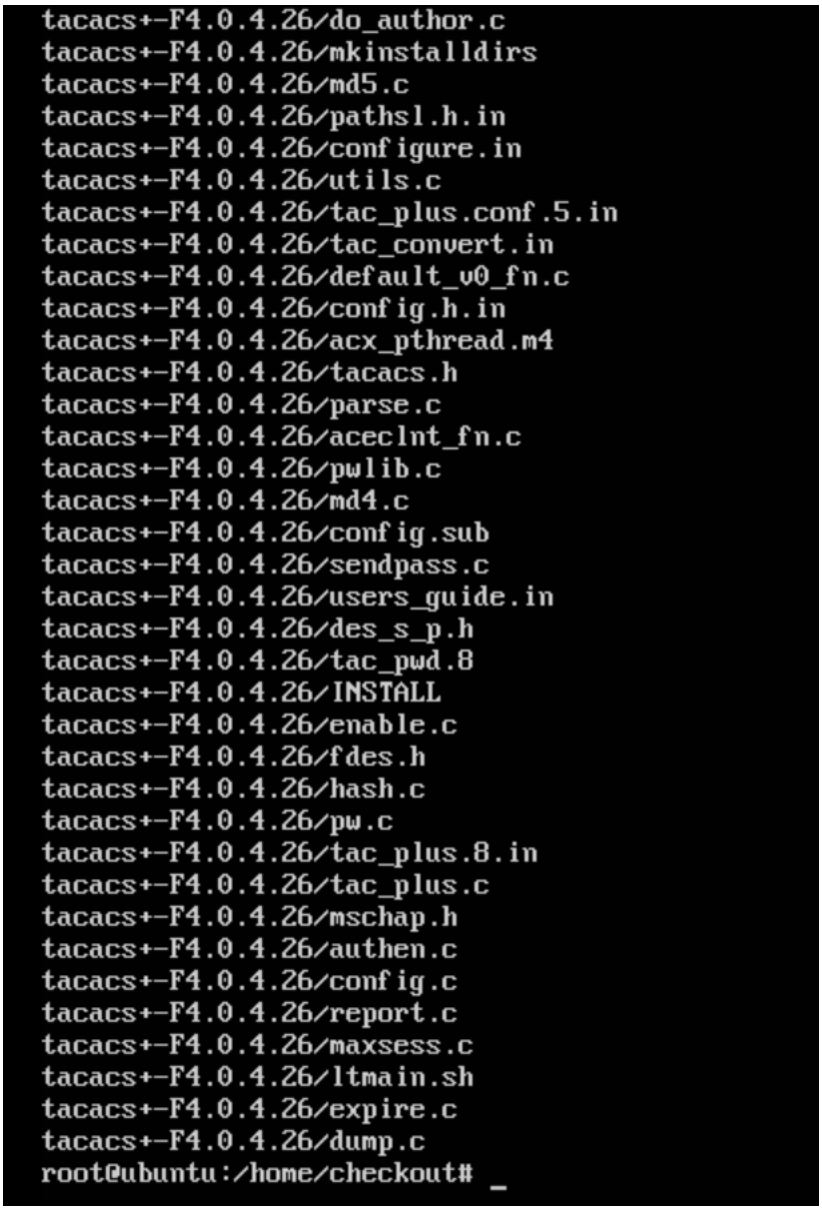


Figure 18: Decompression of the downloaded file

```
root@ubuntu:/home/checkout/tacacs+-F4.0.4.26# less INSTALL
This is a modified version of Cisco's tacacs+ (tac_plus) "developer's
kit."

Quick Installation Guide (an example):

1) ./configure [--prefix=<basedir>]
   By default, All tac_plus crud will be installed under /usr/local.
   This can be overridden with the --prefix option.  E.g.:

        ./configure --prefix=/usr/pkg

   see ./configure --help for other configure options.

2) make install

3) it may be necessary, or you may want to, add tacacs to your /etc/services
   file in order for tacacs to work properly.  eg:

   tacacs       tcp/49

4) read the tac_plug(8) manual page

5) Send any bugs, suggestions or updates to tac_plus@shrubbery.net.
   See the web page at http://www.shrubbery.net/tac_plus.

Prerequisites for building:
- Wietse Venema's TCP wrappers library
INSTALL (END)
```

Figure 19: Installation instructions

Q10: What does the–prefix option do?
A10: It changes the installation directory.

Next, we comment out the lines specified in `packet.c`.

```
    /* get memory for the packet */
    len = TAC_PLUS_HDR_SIZE + ntohl(hdr.datalength);
    //if ((ntohl(hdr.datalength) & ~0xffffUL) ||
//      (len < TAC_PLUS_HDR_SIZE) || (len > 0x10000)) {
//      report(LOG_ERR, "%s: Illegal data size: %lu\n", session.peer,
//             (unsigned long)ntohl(hdr.datalength));
//      return(NULL);
  //  }
    pkt = (u_char *)tac_malloc(len);

    /* initialise the packet */
    memcpy(pkt, &hdr, TAC_PLUS_HDR_SIZE);

    /* the data start here */
    data = pkt + TAC_PLUS_HDR_SIZE;
```

Figure 20: Code commented out

Next, we navigate to the directory and run `./configure -help` to see the

installation options for tac_plus.

Figure 21: config help

Then, we run the installation command.

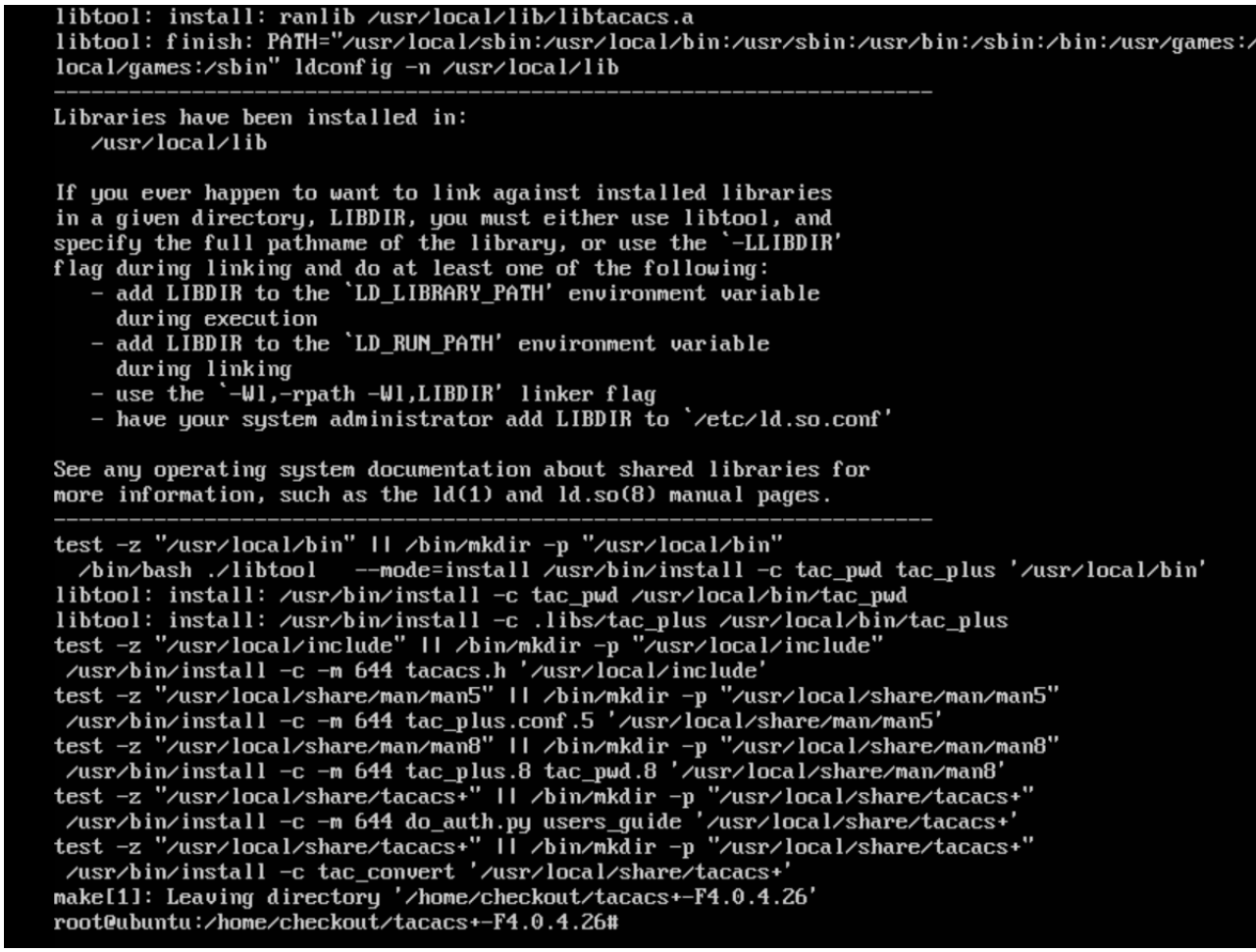


Figure 22: Running the installation

Then, we edit the /etc/ld.so.conf file.

Figure 23

Next, we reload the libraries with the `ldconfig` command.

3.1.4.2 Step 2:

We begin by running the following commands to create the tacacs config file, and then we make a file for the accounting logs.

Then, we edit the config file based on Appendix F, and make it executable.

```
# Encryption key

key = "tac_test"

# Set where to send accounting records

accounting syslog;
accounting file = /var/log/tac_plus/tac_plus.acct

# ACL for network_admin group

acl = network_admin {
        # allow access from all sources
        permit = .*
        # implicit deny (ie: anything else)
}

# ACL for sys_admin group

acl = sys_admin {
        # allow access from 10.10.10.250 only
        permit = ^10\.10\.10\.2$
        # implicit deny (ie: anything else)
}

# network_admin group, full access to network devices

        group = network_admin {
        default service = permit
        expires = "Jan 1 2015"
        acl = network_admin
        service = exec {
         priv-lvl = 15
        }
}

# sys_admin group, only has read access to the network devices and can change the access vl
an on an interface

        group = sys_admin {
        default service = deny
        expires = "Jan 1 2015"
        acl = sys_admin
        service = exec {
        priv-lvl = 0
        }
        cmd = enable {
                permit .*
        }
        cmd = show {
                permit .*
        }
        cmd = exit {
                permit .*
"tac_plus.conf" [readonly] 96L, 1738C                                 1,1           Top
```

Figure 24: Tacacs+ Config File

Next, we make the file for accounting logs in `/var/log/tac_plus`.

Q11: Compare the permissions under each ACL group.

A11: The network_admin group has permission from any address while the sys_admin group only has permission from a particular host.

Q12: Which commands for the sysadmin group are only allowed with certain options?
A12: Some of them are configure, show, interface

Then, we run the `tac_pwd` command, and add the password generated to the config file.

```
checkout@ubuntu:/etc/tacacs$ tac_pwd
Password to be encrypted:
```

Figure 25: Password Generation

```
# User jonathanm using DES password and enable passwords

user = jonathanm {
        member = network_admin
        login = des TCzI3Ng5xqccY
        enable = des Hp.t/nvwy0nZY
}

# User bob authenticating from the system /etc/passwd and the default enable password

user = bob {
        login = file /etc/passwd
        member = sys_admin
}

# Global enable level 15 password

user = $enab15$
        login = des M87fnoeeJCRwI

                                                                96,1
```

Figure 26: Passwords Added to the config file

3.1.4.3 Step 3:

We begin this step by creating a tac_plus executable that con be used to start the service in the `/etc/default` directory.

Next, we copy the following script into `/etc/default/tac_plus`.

```
#!/bin/sh
#
### BEGIN INIT INFO
# Provides: tac-plus
# Required-Start: $network
# Required-Stop:
# Default-Start: 2 3 4 5
# Default-Stop: S 0 1 6
# Short-Description: Start tac-plus server.
# Description: Run the tac-plus server listening for
# AAA ( access, acounting and autorization request )
# from routers or RAS (remote access servers) via
# tacacs+ protocol
### END INIT INFO
PATH=/sbin:/bin:/usr/sbin:/usr/bin
DAEMON=/usr/bin/tac_plus
NAME=tac_plus
DESC="Tacacs+ server"
OTHER_OPTS="-d 256" # Default, if no /etc/default/tac-plus availabl

CONFIG_FILE="/etc/tacacs/tac_plus.conf" # Default, if no /etc/defau

test -f $DAEMON || exit 0
if [ -r /etc/default/tac_plus ] ; then
 . /etc/default/tac_plus
fi
DAEMON_OPTS="-C $CONFIG_FILE $OTHER_OPTS"
case "$1" in
 start)
 echo -n "Starting $DESC: "
start-stop-daemon --start --quiet --pidfile /var/run/$NAME.pid --ex
 echo "$NAME."
 ;;
 stop)
 echo -n "Stopping $DESC: "
start-stop-daemon --stop --quiet --pidfile /var/run/$NAME.pid --exe
 echo "$NAME."
 ;;
 *)
 N=/etc/init.d/$NAME
 echo "Usage: $N {start|stop}" >&2
 exit 1
 ;;
esac
exit 0
```

Figure 27: The start/stop script for tacacs

Then we start the daemon with `/etc/init.d/tac_plus start`, and verify that it is running with the following commands:

```
checkout@ubuntu:~$ ps aux | grep tac_plus
root      13446  0.0  0.0  21896     4 ?        S    19:10   0:00 /usr/bin/tac_plus -C /etc/tacacs/tac_plus.conf -d 16 -L
checkout  14677  0.0  0.3  14224   736 pts/2    S+   21:57   0:00 grep --color=auto tac_plus
```

Figure 28: tac_plus process

```
checkout@ubuntu:~$ netstat -an | grep :49
tcp        0      0 0.0.0.0:49              0.0.0.0:*               LISTEN
```

Figure 29: Tacacs listens on port 49

3.1.5 Step 3:

3.1.6 Key Learning/Takeaways:

This exercise gave us an overall introduction to the TACACS+ protocol. In particular, we learned that there are two main pieces of software to implement the protocol. These are the TACACS.net program available for windows, and the tac_plus daemon available for linux. Upon studying the features of each, it is apparent that the windows version is easier to install, and comes with more configuration files and tools to make the TACACS+ protocol more variable. The tac_plus daemon take a bit more effort to install and configure so that it works correctly, but is much easier to manage as we have easy access to the source code. Because of these facts, we continue the rest of the lab working with the tac_plus daemon. Working with the protocol in this exercise made us appreciate the abilities of TACACS+. An example of a key takeaway from this exercise included learning how the exchange of packets between a device and a TACACS+ server worked, namely separating all aspects of AAA to make the protocol more flexible.

3.2 Exercise 2 - Setting up GNS3 for use with TACACS+ (Troy)

3.2.1 Analysis & Evidence:

3.2.1.1 Step 1: Introduction

We began this exercise by setting up a host-only network on vmnet3, as specified in the lab instructions. As seen in figure 30 below, this contained a subnet IP of 10.10.10.0, with a netmask of /24, or 255.255.255.0.

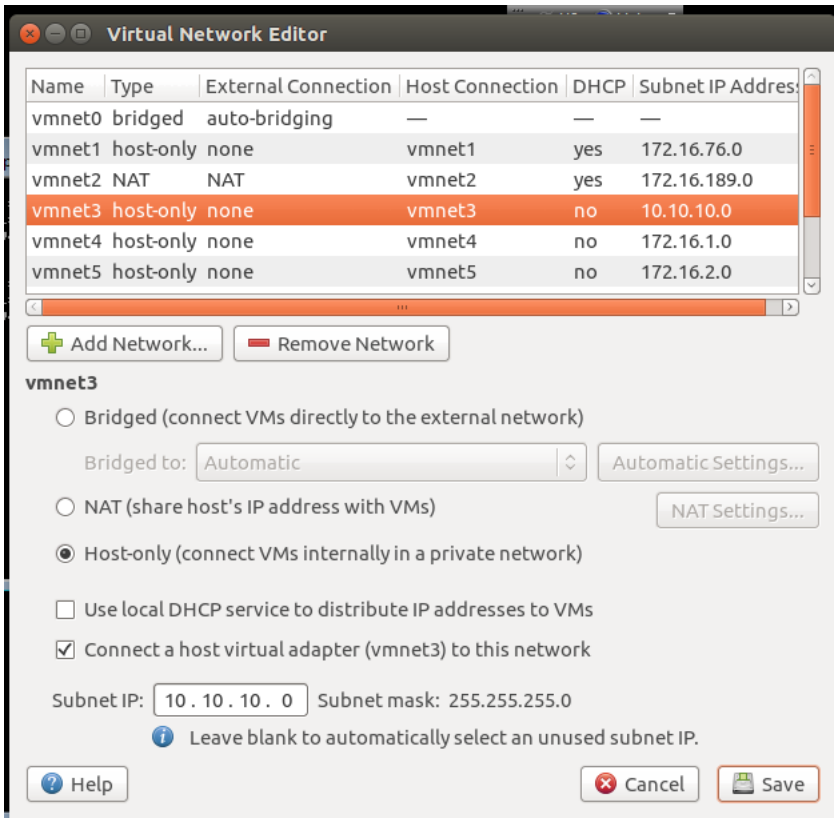


Figure 30: setting up vmnet3

Then, we installed the GNS3 software, as seen in figure 31. Figure 31 also shows the running of the software at the bottom half of the terminal.

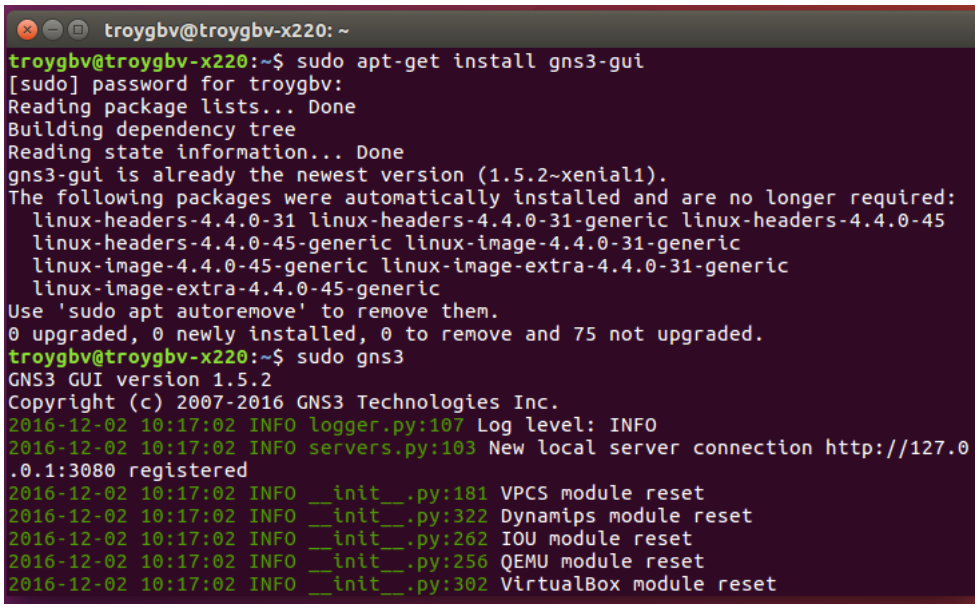


Figure 31: installing gns3

Q1:- What does GNS3 stand for, and what is it primarily used for?
A1:GNS3 stands for "Graphical Network Simulator 3". It is primarily used to use virtual and real network technologies and developing "real-world" networks in a private environment.

Once gns3 was open, we installed and added the cisco 3600 router to our topology, as seen in figure 32. This figure also shows the execution of the `show run` command (once the router was started and consoled into) to show part of the router's current running configuration.

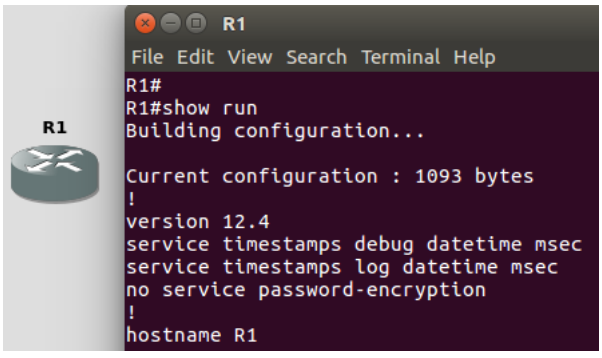


Figure 32: adding the cisco router

Then, we set a static an IP on the router, giving it an IP address of

10.10.10.2/24, as seen in figure 33, 34, and 35

```
R1#show ip interface brief
Interface                  IP-Address      OK? Method Status                Prot
ocol
Ethernet0/0                unassigned      YES unset  administratively down down

FastEthernet1/0            unassigned      YES unset  administratively down down

Ethernet2/0                unassigned      YES unset  administratively down down

Ethernet2/1                unassigned      YES unset  administratively down down

Ethernet2/2                unassigned      YES unset  administratively down down

Ethernet2/3                unassigned      YES unset  administratively down down

R1#
```

Figure 33: showing the available network on the router

Q2: What do the previous commands show ... do?
A2: `show run` shows the current running configuration of the cisco router. This command is commonly used after a configuration has been set, to make sure that it was validated.`show ip interface brief` shows the network interfaces available on the router, asa well as a brief description of them (name, assigned or not, status, etc.)

Q3: In terms of cisco iOS syntax, what is the difference between the ">" prompt and the "#" prompt?
A3: The ">" prompt refers to the "user exec" mode, while the "#" refers to the "privilege exec" mode, also known as the "enable" mode.

```
R1#config t
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#interface FastEthernet1/0
R1(config-if)#ip address 10.10.10.2 255.255.255.0
R1(config-if)#no shutdown
R1(config-if)#
*Mar  1 00:01:40.247: %LINK-3-UPDOWN: Interface FastEthernet1/0, changed state t
o up
*Mar  1 00:01:41.247: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthern
et1/0, changed state to up
R1(config-if)#end
R1#
*Mar  1 00:01:42.991: %SYS-5-CONFIG_I: Configured from console by console
R1#
```

Figure 34: commands to set the static address

```
R1#show ip interface brief
Interface                  IP-Address      OK? Method Status                Prot
ocol
Ethernet0/0                unassigned      YES unset  administratively down down

FastEthernet1/0            10.10.10.2      YES manual up                    up
```

Figure 35: showing the configured interface, FastEthernet1/0

Once the router was configured, we then added an Ubuntu Server VM to the topology to act as the TACACS+ server that the router looks to to provide its AAA services. The preliminary topology can be seen in figure 36 below.

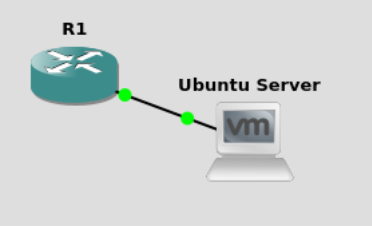


Figure 36: Adding a server vm to the gns3 topology

Q4: Did you set the vm to have two network adapters and allow gns3 to use it? Show evidence.

A4: Yes, see figure 38.

We connected the newly added vm to the vmnet3 network (through a new network adapter) and gave it a static ip address of 10.10.10.100/24. See figure 37.

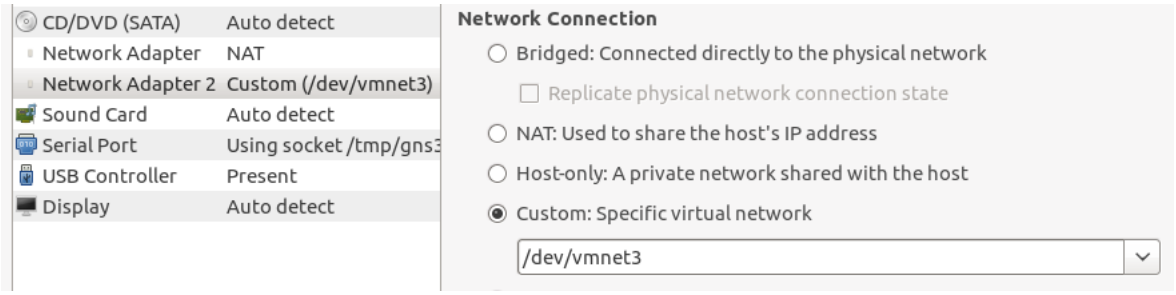


Figure 37: adding server to vmnet3 on vmware

To make sure that the right network interface was connected on the server, we had to configure the server through gns3 to allow it to "use any configured VMware adapter" and change the number of network interfaces to 2. See figure 38.

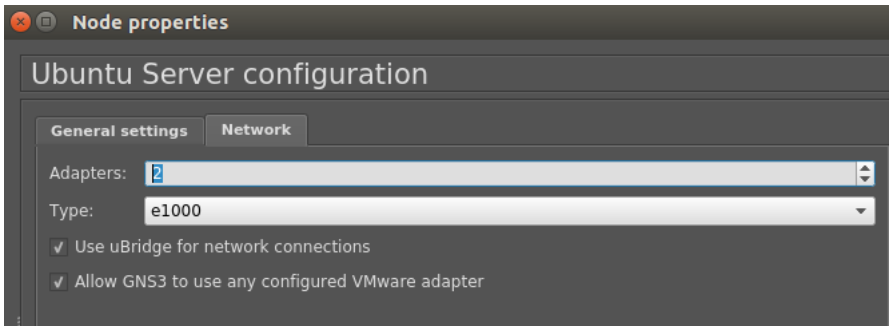


Figure 38: configuring server vm on gns3

We then started a wireshark capture on the router through gns3. To verify that the connection was successful, we captured two conversations; pinging the server vm from the router, and vice versa. An example of this can be seen in the figures 39 and 40.

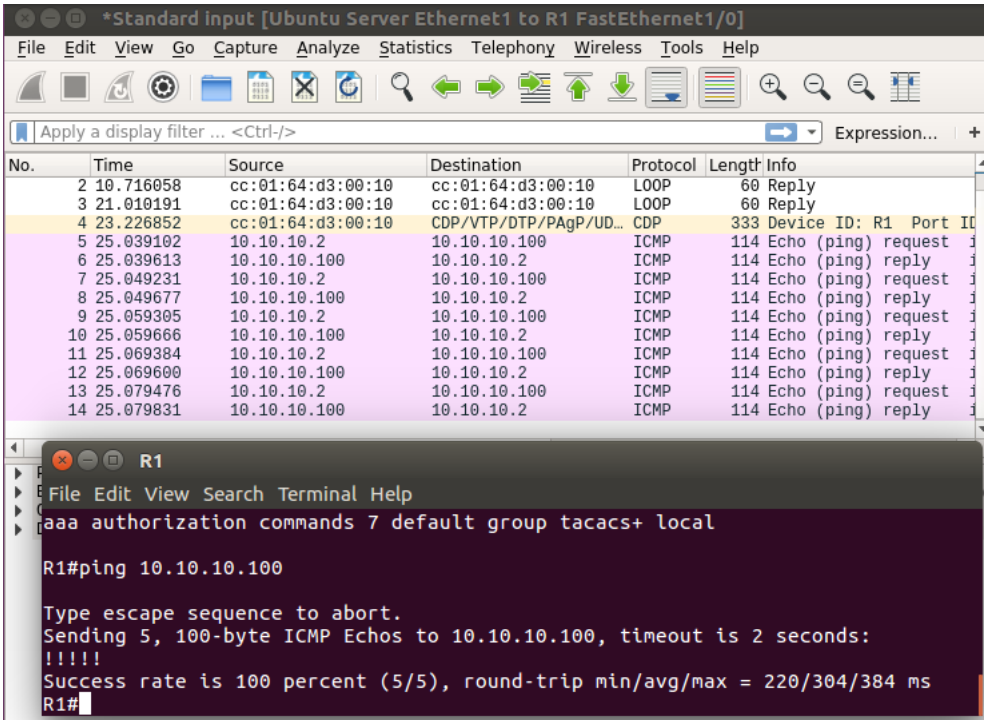


Figure 39: pinging the server (10.10.10.100) from the router (10.10.10.2)

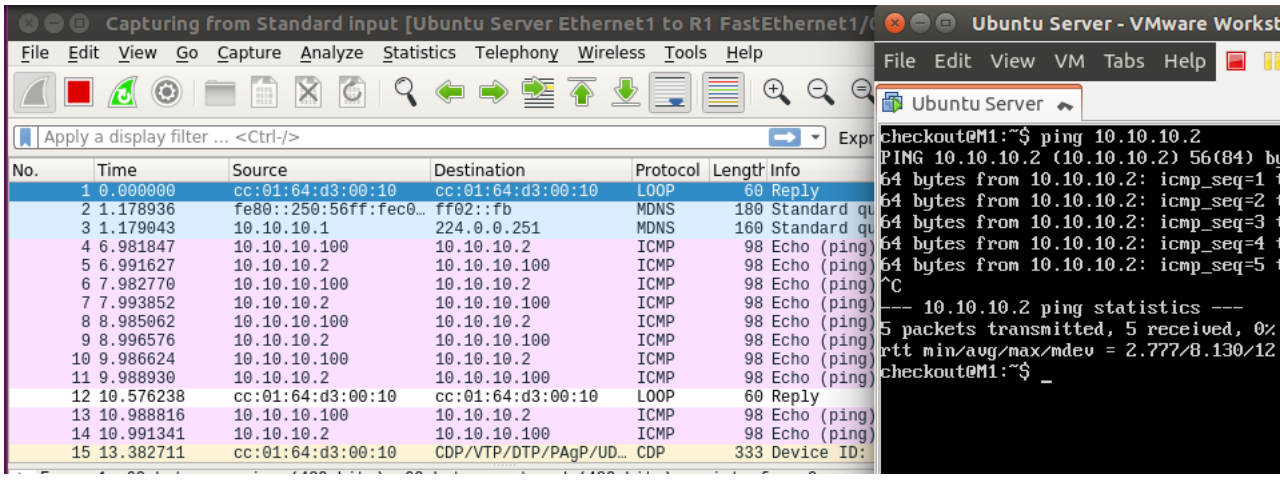


Figure 40: pinging the router (10.10.10.2) from the server (10.10.10.100)

These captures in figures 39 and 40 can be seen in the following wireshark capture files attached: "R_to _S _icmp.pcapng" and "S_to _R _icmp.pcapng".

Q5: Did you perform the ping successfully on both machines? Provide evidence.

A5: Yes, see figures 39 and 40.

Q6: What are the MAC addresses relating to the VM and the router?

A6: The MAC address of the Router is cc:01:64:d3:00:10. The MAC address of the Server is 00:0c:29:1c:73:9f.

3.2.1.2 Step 2: Testing tacacs server on preliminary topology

Now that the preliminary topology is set up , we were then required to input a list of commands into the router console, so that the router would have the correct configuration to utilize the TACACS+ services. See figure 41. Figure 42 shows the `show run` command on the router to verify that these configuration commands have taken place.

```
R1(config)#tacacs-server host 10.10.10.100
R1(config)#tacacs-server directed-request
R1(config)#tacacs-server key tac_test
R1(config)#aaa new-model
R1(config)#aaa authentication login default group tacacs+ local
R1(config)#aaa authentication enable default group tacacs+ enable
R1(config)#aaa authorization config-commands
R1(config)#aaa authorization commands 0 default group tacacs+ local
R1(config)#aaa authorization commands 1 default group tacacs+ local
R1(config)#aaa authorization commands 7 default group tacacs+ local
R1(config)#aaa authorization commands 15 default group tacacs+ local
R1(config)#aaa accounting commands 0 default start-stop group tacacs+
R1(config)#aaa accounting commands 1 default start-stop group tacacs+
R1(config)#aaa accounting commands 7 default start-stop group tacacs+
R1(config)#aaa accounting commands 15 default start-stop group tacacs+
R1(config)#aaa accounting network 0 start-stop group tacacs+
R1(config)#aaa accounting network 15 start-stop group tacacs+
R1(config)#aaa accounting connection 0 start-stop group tacacs+
R1(config)#aaa accounting connection 15 start-stop group tacacs+
R1(config)#aaa session-id common
R1(config)#
```

Figure 41: inputting tacacs+ configuration commands on the router

```
R1#show run
Building configuration...

Current configuration : 2033 bytes
!
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname R1
!
boot-start-marker
boot-end-marker
!
!
aaa new-model
!
!
aaa authentication login default group tacacs+ local
aaa authentication enable default group tacacs+ enable
aaa authorization config-commands
aaa authorization commands 0 default group tacacs+ local
aaa authorization commands 1 default group tacacs+ local
aaa authorization commands 7 default group tacacs+ local
 --More--
```

Figure 42: verification of the commands in figure 41

Q7: What is the purpose of inputting all of these commands? In particular, what do the commands that have "local" at the end specify? What do the numbers "0, 1, 7, and 15" mean?
A7: The list of commands as shown in figure 41 are all required to enable

the TACACS+ services on the router. The commands that have "local" at the end tell the router that if no TACACS+ server responds, look to the local account specified.(if no such local account exists, deny all forms of any attempts to log in) The numbers "0, 1, 7, and 15" refer to the different privilege numbers for a user utilizing the TACACS+ service.

Now, we added a client to access the router, through means of an Ubuntu Desktop VM. We also connected the desktop vm to vmnet3, and assigned it a static IP address of 10.10.10.101/24. On the gns3 topology, we added a switch to the middle, and connected all of the machines, so it looks like figure 43.

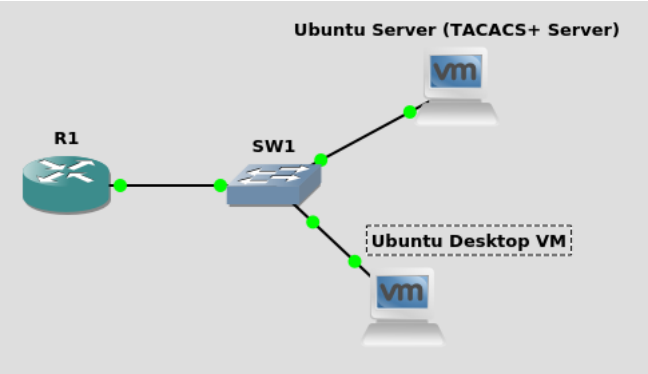


Figure 43

We then started a wireshark capture on the Ubuntu Server vm and prepared to access the router from the client (Ubuntu Desktop VM). In doing so, we tailed the two log files that the tacacs+ server uses for AAA services. These tails can be seen in the figures 44 and 45.

```
heckout@M1:~$ tail -f /var/log/tac_plus.log
Mon Dec  5 11:53:06 2016 [1490]: connect from 10.10.10.2 [10.10.10.2]
Mon Dec  5 11:53:07 2016 [1489]: enable query for 'test' tty130 from 10.10.10.2 accepted
Mon Dec  5 11:53:10 2016 [1290]: session.peerip is 10.10.10.2
Mon Dec  5 11:53:10 2016 [1491]: connect from 10.10.10.2 [10.10.10.2]
Mon Dec  5 11:53:10 2016 [1491]: cfg_acl_check(network_admin, 10.10.10.2)
Mon Dec  5 11:53:10 2016 [1491]: ip 10.10.10.2 matched permit regex .* of acl filter network_a
Mon Dec  5 11:53:10 2016 [1491]: host ACLs for user 'test' permit
Mon Dec  5 11:53:10 2016 [1491]: authorization query for 'test' tty130 from 10.10.10.2 accepte
Mon Dec  5 11:53:10 2016 [1290]: session.peerip is 10.10.10.2
Mon Dec  5 11:53:10 2016 [1492]: connect from 10.10.10.2 [10.10.10.2]
```

Figure 44: tail of the /var/log/tac_plus.log file

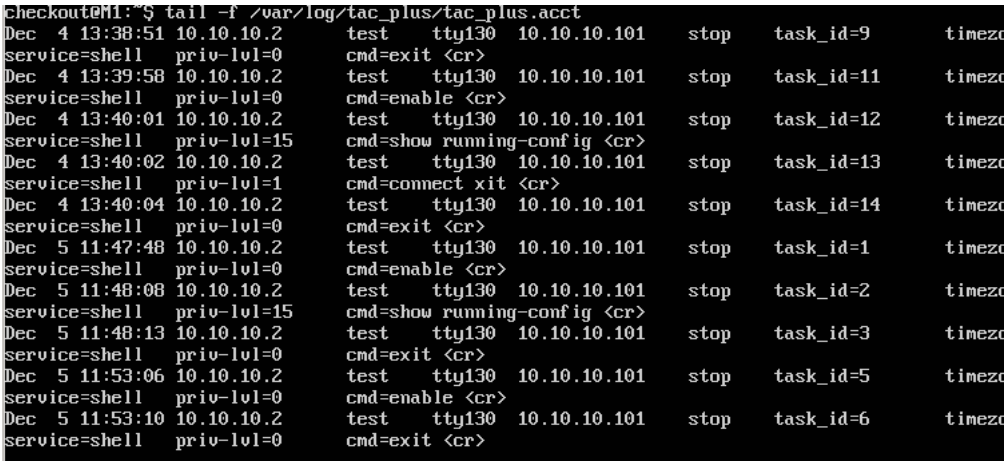


Figure 45: tail of the /var/log/tac_plus/tac_plus.acct file

As you can see in figures 44 and 45, the authorization and authentication parts of the conversation are found in 44 and the accounting parts are found in 45. This proves to be useful, as a network administrator could have these files up at all times to see all the activity that is occuring when a user attempts to authenticate to the tacacs+ server.

Q8: What is the difference between the two log files that are being tailed? What are the purposes of them?

A8: The "tac_plus.log" file is the main log that the TACACS+ server stores all activity regarding the authorization and authentication packets coming through. The "tac_plus _acct" file stores all the information in regards to accounting services.

In figure 46, we then restart the tacacs+ server and check the status with the `sudo service` commands.

```
checkout@M1:~$ sudo /etc/init.d/tac_plus stop
Stopping Tacacs+ server: tac_plus.
checkout@M1:~$ sudo /etc/init.d/tac_plus start
Starting Tacacs+ server: tac_plus.
checkout@M1:~$ sudo service tac_plus restart
checkout@M1:~$ sudo service tac_plus status
• tac_plus.service - LSB: Start tac-plus server.
   Loaded: loaded (/etc/init.d/tac_plus; bad; vendor preset: enabled)
   Active: active (running) since Mon 2016-12-05 12:14:53 PST; 2s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 1721 ExecStop=/etc/init.d/tac_plus stop (code=exited, status=0/SUCCESS)
  Process: 1725 ExecStart=/etc/init.d/tac_plus start (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/tac_plus.service
           └─1731 /usr/bin/tac_plus -C /etc/tacacs/tac_plus.conf -d 16 -L

Dec 05 12:14:53 M1 systemd[1]: Starting LSB: Start tac-plus server....
Dec 05 12:14:53 M1 tac_plus[1727]: Reading config
Dec 05 12:14:53 M1 tac_plus[1727]: Version F4.0.4.26 Initialized 1
Dec 05 12:14:53 M1 tac_plus[1727]: tac_plus server F4.0.4.26 starting
Dec 05 12:14:53 M1 tac_plus[1725]: Starting Tacacs+ server: tac_plus.
Dec 05 12:14:53 M1 tac_plus[1730]: Backgrounded
Dec 05 12:14:53 M1 tac_plus[1731]: uid=0 euid=0 gid=0 egid=0 s=0
Dec 05 12:14:53 M1 systemd[1]: Started LSB: Start tac-plus server..
checkout@M1:~$
```

Figure 46: Restart of the tacacs+ server

With wireshark still running, we tested the services provided by TACACS+ by telnetting into the router from the ubuntu Desktop VM. By entering the credentials set for a test user (specified in setting up the tac_plus daemon) we have full remote access to the router to configure as need be. Proof of the wirshark capture and the telnet connection can be seen in figure 47.

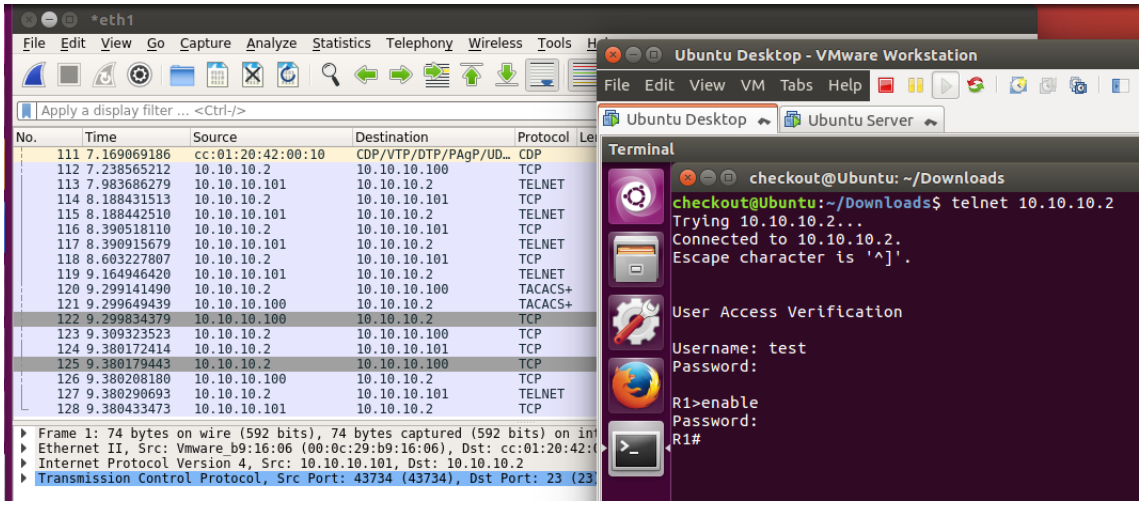


Figure 47: shows the wireshark capture on the server vm while using telnet to connect to tacacs+ router

The packets captured in this conversation can be found as "tacacs_telnet

_1.pcapng" attached. Figure 47 shows the logging into the router, as well as executing the `enable` command (with the enable password set in the configuration files) to have further access to the router. For example, we then ran two commands in the telnet console: `show run` and `show ip in br`. This can be seen in figure 48

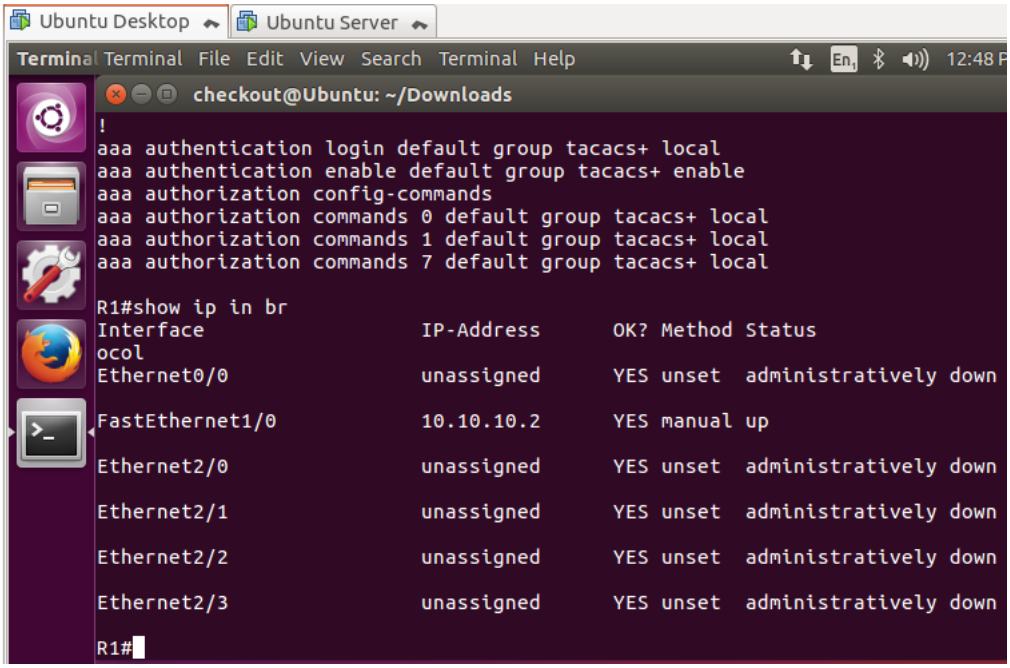


Figure 48: test commands to prove that we have telnet access to the TACACS+ router, from an Ubuntu Desktop Client.

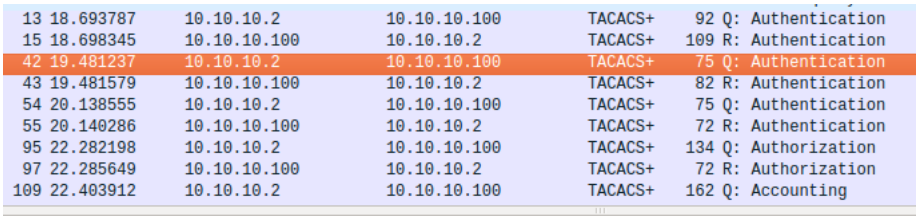


Figure 49: Shows proof of all the aspects of AAA in the telnet capture.

Q9: Can you see all parts of the AAA conversation? Show proof.
A9: Yes, see figure 49.
Q10: What details are made available in the tacacs+ packet headers in plain text?

A10: The Major/Minor version, type of packet (AAA), Sequence number, Session ID, and packet length are all available in plain text on a TACACS+ packet. The rest of the information of the packet is encrypted under "Encrypted Data"

3.2.2 Key Learning/Takeaways:

In this exercise, we learned that GNS3 provided a straight-forward way to create and test a "real-world" network environment with real Cisco device images. Using TCP port 49, TACACS+ Authentication, Authorization, and Accounting services log who has logged in to the device using the service, as well as when and what commands they issued. This can be useful for network administrators if they were needed to account for services in a billing environment, or the perform an audit for security purposes. A main takeaway from this exercise is that through configuration, TACACS+ has the ability to fallback onto a local account, if the router does not receive a response from the specified server.

3.3 Exercise 3: Denial-of-Service attack on TACACS+ with use of Scapy. (Isaac)

3.3.1 Analysis & Evidence:

3.3.1.1 Step 1: Setup and DOS

We begin by making sure that scapy is installed with `dpkg -s scapy`, and see that it is already installed.

```
[sudo] password for thredzeppelin:
dpkg-query: package 'scapy' is not installed and no information is available
Use dpkg --info (= dpkg-deb --info) to examine archive files,
and dpkg --contents (= dpkg-deb --contents) to list their contents.
```

Figure 50: Checking if Scapy is installed

Then, we create and edit the tacacs_scapy.py file based on Appendix E.

```
1  #! /usr/bin/env python
2
3  # Set log level to benefit from Scapy warnings
4  import logging
5  import random
6  logging.getLogger("scapy").setLevel(1)
7
8  from scapy.all import *
9
10 class Tacacs(Packet):
11         name = "Tacacs+"
12         fields_desc=[ XByteField("major_version", 12),
13                       XByteField("minor_version", 0),
14                       ByteEnumField("type", 1, {1:"Authentication", 2:"A
   uthorization", 3:"Accounting"}),
15                       ByteField("seq_no", 1),
16                       ByteEnumField("flags", 1, {1:"TAC_PLUS_UNENCRYPTED
   _FLAG", 4:"TAC_PLUS_SINGLE_CONNECT_FLAG"}),
17                       IntField("session_id", random.SystemRandom().randi
   nt(1, 4294967295)),
18                       IntField("length", 43)]
19
20 if __name__ == "__main__":
21     interact(mydict=globals(), mybanner="Test add-on v3.14")
~
```

Figure 51: tacacs_scapy.py script

Next, we make the script executable with `chmod 755 tacacs_scapy.py` and run the script. Then, we run the scapy commands and get the following output:

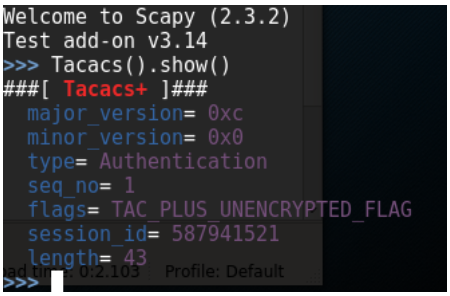


Figure 52: Tacacs+ layer header

Q2: Why did we need the script to examine a Tacacs+ packet with scapy? Hint: Look online for a list of protocol layers that scapyknows by default.
A2: The Tacacs+ layer is not included with scapy by default. The script creates a Tacacs+ layer that we can now interact with and send using scapy.

3.3.1.2 Step 2:

We begin by setting up the Ubuntu Server VM with 512 MB of ram and restarting the VM.

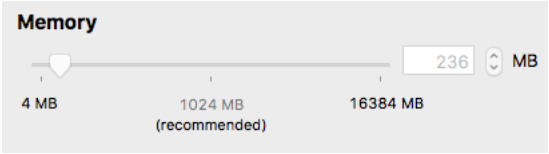


Figure 53: Lowering of Ubuntu Server Ram

Next, we create the text file for the script in Appendix C and make it executable using `chmod`.

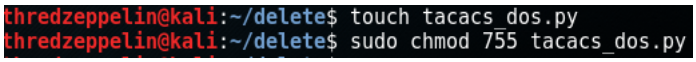


Figure 54: Creating the text file for the script

Then we copy the script into the file.
Q3: What is the script doing? (Give a general answer)
A3: It is simulating the TCP 3-Way-Handshake and sending a packet with a overly-large length field.

Next we open Wireshark on the Ubuntu Server from the host, with the filter set to only display packets using port 49.

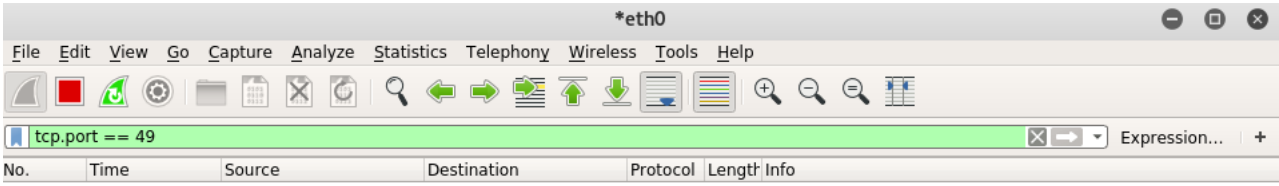


Figure 55: Wireshark with capture filter sent to port 49

Next, we ran the iptables command to drop outgoing RST packets:

```
root@kali:/home/thredzeppelin# iptables -A OUTPUT -p tcp --tcp-flags RST RST -
s 172.16.203.138 -j DROP
```

Figure 56: iptables command

Q4: What does this firewall rule do and why do we need it? (Hint try running the script without it.)
A4: The kernel will send a TCP RST packet if it receives an SYN/ACK destined for a process running in user mode. This tells the firewall to drop the outgoing RST packets.

Then, we run the script, and check Wireshark.

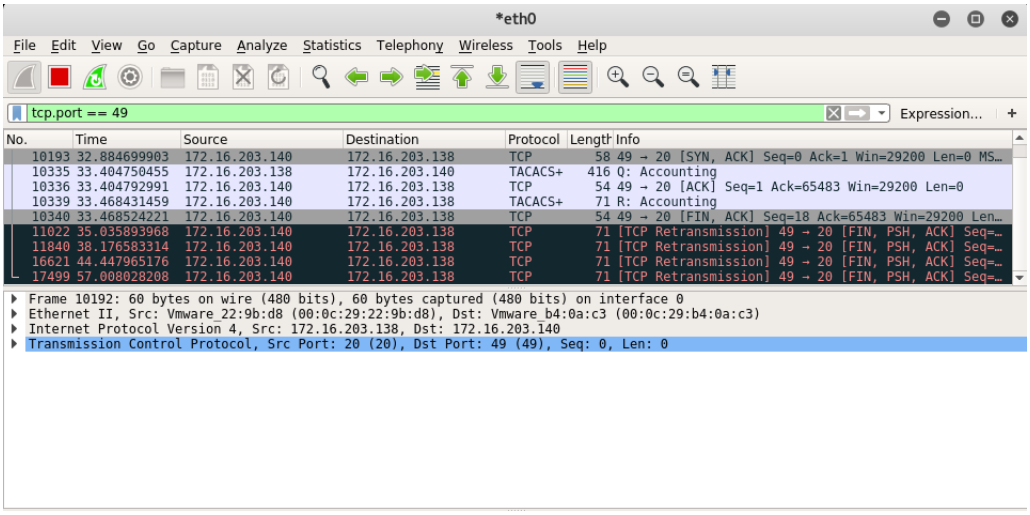


Figure 57: Wireshark after the attack

Then we check on the server if Tacacs+ is still running.

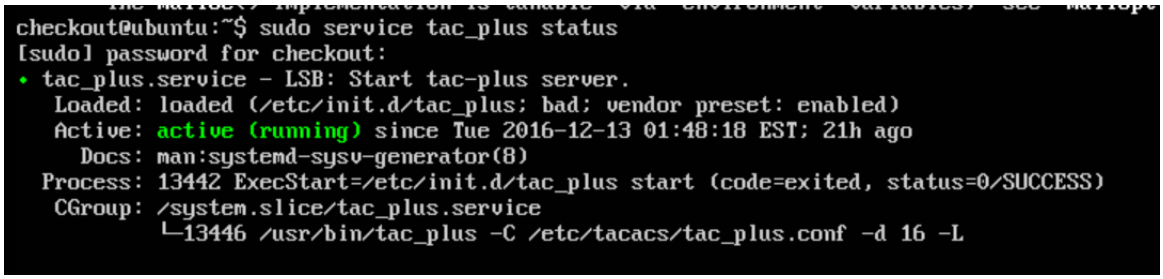


Figure 58: The service is still running

Next, we examine the man page for malloc on the server.

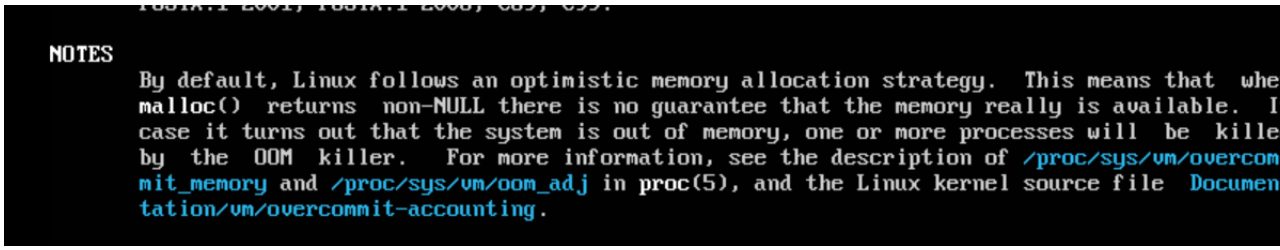


Figure 59: Malloc notes

Then, we copy the script in Appendix D and make it executable like the previous scripts.

Next, we restarted tac_plus and the wireshark capture and executed the attack.

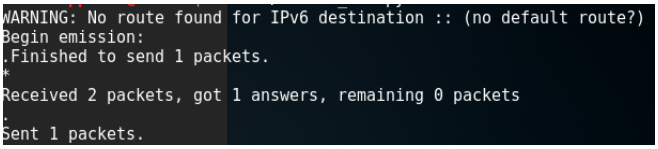


Figure 60: Output from the script

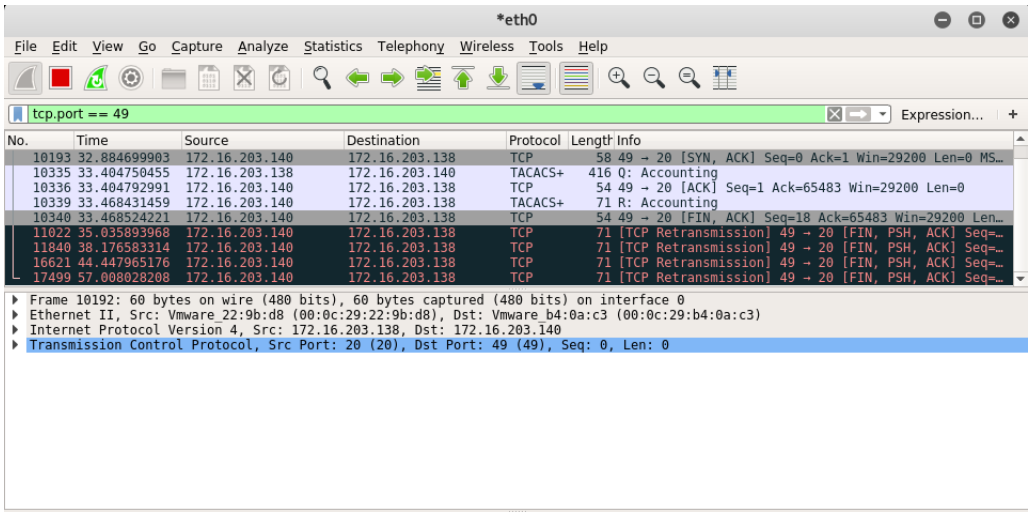


Figure 61: Packets captured from updated attack

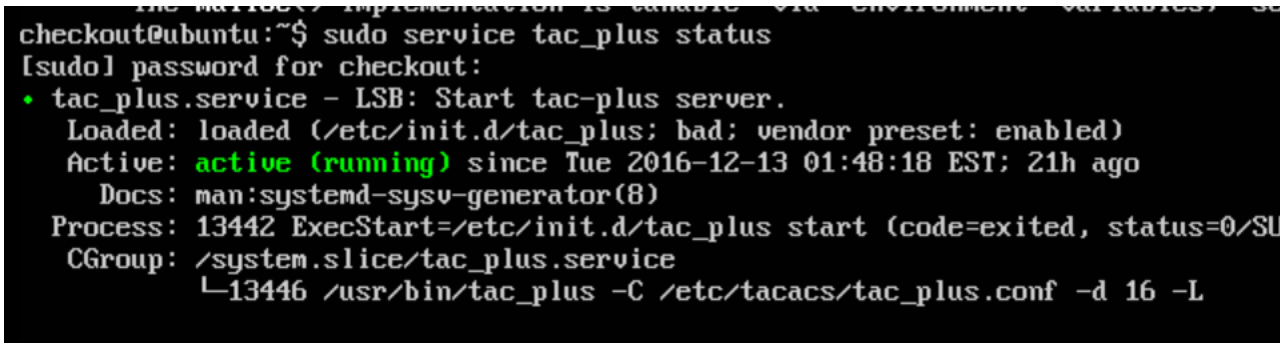


Figure 62: Tacacs+ is still running

Q5: Why did the attack fail this time?
A5: There was not a large enough payload for the attack to overload the server. This limiting factor is due to the implementation of scapy.

3.3.2 Key Learning/Takeaways:

In this exercise we learned that when implementing an attack, it is very important to choose the correct tools. Scapy wasn't designed to send to send a packet with an outrageous payload. I imagine an attacker would have to write software from scratch to implement this attack successfully. I also believe it would be difficult to find a version of tacacs that hasn't been updated since the late 1990s. That being said, I'm sure this vulnerable version is running out there somewhere.

3.4 Exercise 4: TACACS+ attack by use of Man in The Middle (Taco Taco) (Troy)

3.4.1 Analysis & Evidence:

3.4.1.1 Step 1: Introduction and setup

We first set up an environment that was similar to the one in Exercise 2, but replacing the Ubuntu Desktop Client in GNS3 with a Kali Linux VM. We set Kali Linux to have a static IP address of 10.10.10.6/24 on eth1, and linked in to the vmnet3-host-only network that we specified in exercise 2. See figures 63 and 64 below.

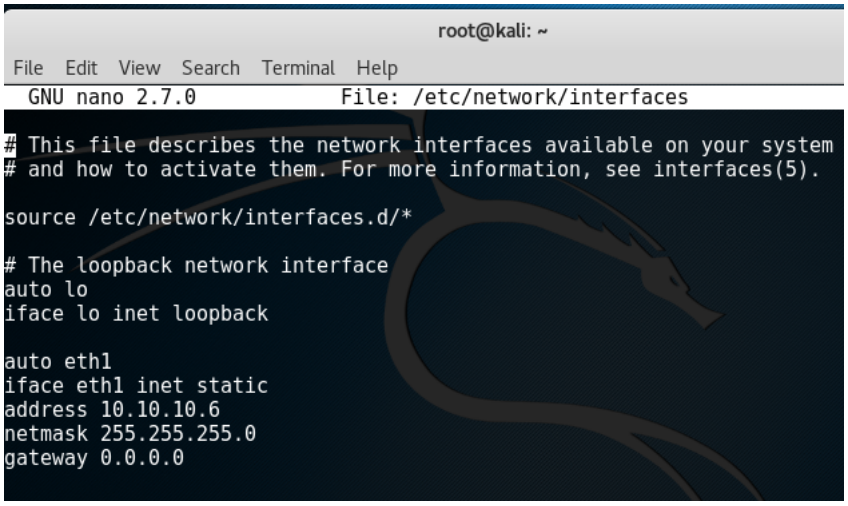


Figure 63: Setting a static IP on Kali Linux VM

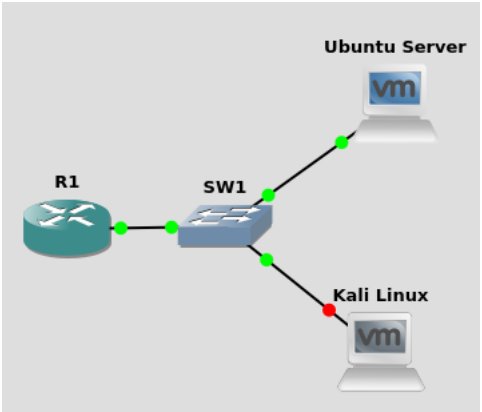


Figure 64: GNS3 topology of Exercise 4

We also verified that the Kali Linux VM was able to ping the other machines in the topology (Router and Server). See figure 65

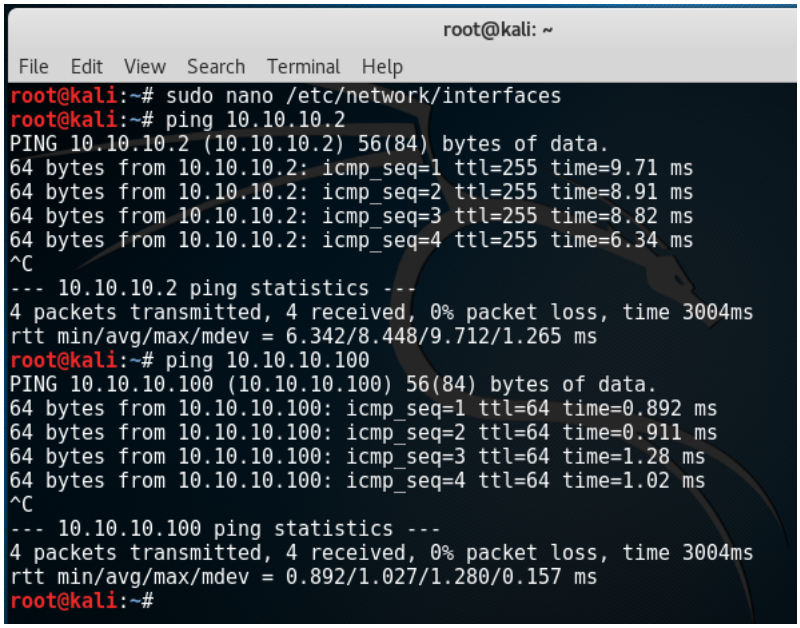


Figure 65: Issue of pings to the TACACS+ router (10.10.10.2) and the TACACS+ server (10.10.10.100) from Kali Linux VM

After, we downloaded the TacoTaco project off of github, and extracted the contents to our Downloads folder on the Kali Linux VM. See figure 66 for proof.

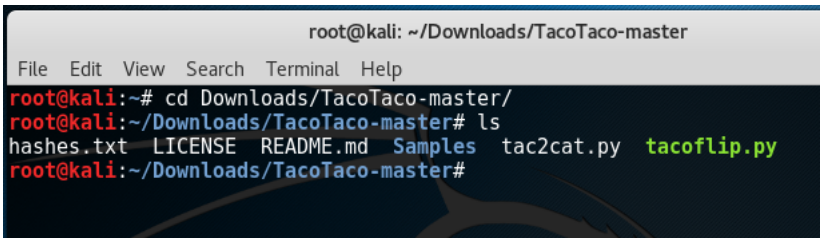


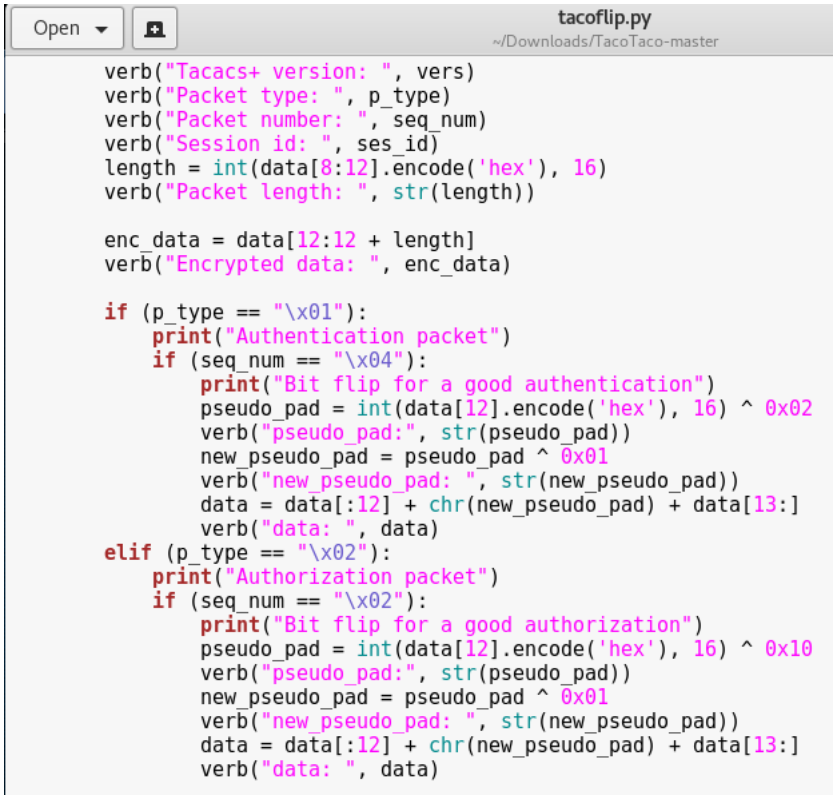
Figure 66: Shows the extracted TacoTaco project to the Downloads folder in Kali

Q1: How does TACACS+ encrypt the entire body of the packet? Reference the RFCs here.
A1: The entire body of a TACACS+ packet is encrypted by "XORing" the packet body data (byte-wise) with a "pseudo-random" pad that the TACACS+ protocol defines. This XOR function looks like this: `Encrypted`

`(data) == data ^ pseudo_pad`. This pad is made by joining a series of MD5 hashes (of 16 bytes) and truncating it to the length of the original data. The pad looks like this: `pseudo_pad = (MD5_1, MD5_2,...MD5_n) truc. to len(data)`. Each MD5 hash is composed of the data in headers seen in the packet (made available in plain text), and the Pre-Shared-Key. The hash looks like this: `MD5_n = MD5(session id, pre-shared-key, version, sequence number, and MD5_n-1)`. [1]

Q2: How are we able to perform the Taco-Taco tacoflip attack with use of Man-in-the-middle? In particular, what is the specific vulnerability that we are taking advantage of?
A2: In the Authentication and Authorization part of TACACS+, the first byte of the reply determines whether the user has been granted access to the device or not (0x01 for acceptance, 0x02 for deny). Taking advantage of the fact that TACACS+ does not integrate any form of integrity checks, we are able to use the TacoTaco attack to change the first byte of TACACS+ packets without the server knowing that it was maliciously changed. This is how the Man-in-the-middle attack is achieved. [2]

Q3: Examine the tacoflip.py file in a text editor of your choice. What exactly is it doing?
A3: See figure 67. The tacoflip python file takes advantage of the fact that we know how the TACACS+ protocol is structured by use of a "bit flipping" attack. It gets the "pseudo-pad" that TACACS+ makes by "XORing" the encrypted first byte of the packet and the decrypted byte. The decrypted byte is known since the byte returns as "0x02" if the user is denied access. The python file then "XORs" the newly obtained "pseudo-pad" with the byte relating to successful authentication (0x01), puts this XOR'ed byte into the encrypted packet, and sends it to the server, all during transmission.




```
        verb("Tacacs+ version: ", vers)
        verb("Packet type: ", p_type)
        verb("Packet number: ", seq_num)
        verb("Session id: ", ses_id)
        length = int(data[8:12].encode('hex'), 16)
        verb("Packet length: ", str(length))

        enc_data = data[12:12 + length]
        verb("Encrypted data: ", enc_data)

        if (p_type == "\x01"):
            print("Authentication packet")
            if (seq_num == "\x04"):
                print("Bit flip for a good authentication")
                pseudo_pad = int(data[12].encode('hex'), 16) ^ 0x02
                verb("pseudo_pad:", str(pseudo_pad))
                new_pseudo_pad = pseudo_pad ^ 0x01
                verb("new_pseudo_pad: ", str(new_pseudo_pad))
                data = data[:12] + chr(new_pseudo_pad) + data[13:]
                verb("data: ", data)
        elif (p_type == "\x02"):
            print("Authorization packet")
            if (seq_num == "\x02"):
                print("Bit flip for a good authorization")
                pseudo_pad = int(data[12].encode('hex'), 16) ^ 0x10
                verb("pseudo_pad:", str(pseudo_pad))
                new_pseudo_pad = pseudo_pad ^ 0x01
                verb("new_pseudo_pad: ", str(new_pseudo_pad))
                data = data[:12] + chr(new_pseudo_pad) + data[13:]
                verb("data: ", data)
```

Figure 67: Part of the expansion of the tacoflip.py file in gedit. Here you can see the process of how the script carries out the bitflipping attack as explained in Question 3.

Q4: What is arp spoofing / poisoning? Why do we need it alongside the attack?

A4: Arp spoofing is a type of attack in which a machine sends false ARP messages over a network. This Man-in-the-middle attack allows the attacker's illegitimate MAC address to be linked to a legitimate IP address on the network. This is needed alongside the attack because ARP spoofing tricks the TACACS+ configured device (in our case the router) that the Attacker's machine has the same MAC and IP address as the TACACS+ server. This allows us to send, receive, and alter all traffic meant for the server on the attacker's machine. [3]

Then, we set up the cisco router to accept ssh instead of telnet, and verified the version, and whether or not it was up and running.

Q5: Did you set up ssh to the router and verify that it is active?

A5: Yes, the configuration commands and verification are seen in figure 68. below.

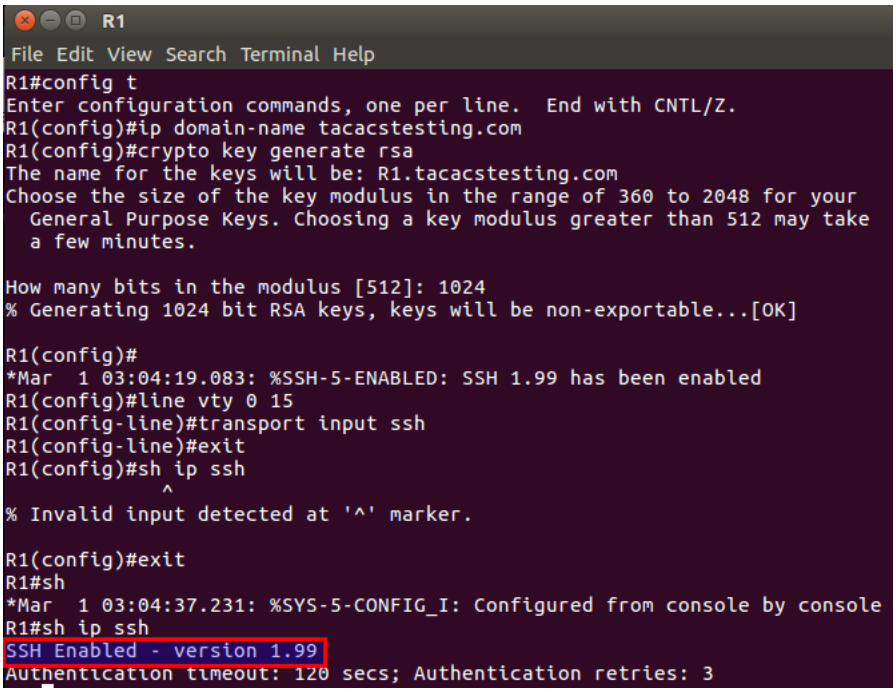


Figure 68: Setting up SSH in the console of the cisco router. Here, you can see that the version is 1.99, and that the keys were generated under the created domain name of "tacacstesting.com"

3.4.1.2 Step 2:Initiating the tacoflip.py attack

To initiate the tacoflip.py attack, we first opened wireshark on Kali Linux and set it to capture packets on the "eth1" interface, the one we manually configured earlier. We then ran the tacoflip.py file, as seen in figure 69 below.

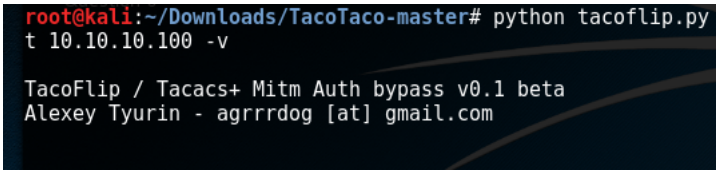


Figure 69: Output of running the tacoflip.py file initially.

Q7: How can you figure out what the "-t" and "-v" options do?
A7: See figure 70. The -t is for "target" and the -v is for "verbose. This

can be seen upon expansion of the python script.

```
def parse_args():
    parser = argparse.ArgumentParser(
        description="A tool for authentication/authorization bypass by MitM attack on Cisco
devices and a Tacacs+ server")
    parser.add_argument(
        '-t', '--target', type=str, help='An IP address of a  Tacacs+ server', required=True)
    parser.add_argument(
        '-v', '--verbose', help='Verbose mode', action="store_true", dest="verbose", default=False,
required=False)
    args = parser.parse_args()
    return args.target, args.verbose
```

Figure 70: Shows the beginning of the expansion of the tacoflip.py file, showing the available options.

Next, we used **arpspoof** on the eth1 interface, telling the TACACS+ router that the Kali Linux attacker has the same MAC address as the TACACS+ server. See figure 71.

```
root@kali:~# arpspoof -i eth1 -t 10.10.10.100 10.10.10.2
0:c:29:54:33:4b 0:c:29:1c:73:9f 0806 42: arp reply 10.10.10.2 is-at 0:c:2
9:54:33:4b
0:c:29:54:33:4b 0:c:29:1c:73:9f 0806 42: arp reply 10.10.10.2 is-at 0:c:2
9:54:33:4b
0:c:29:54:33:4b 0:c:29:1c:73:9f 0806 42: arp reply 10.10.10.2 is-at 0:c:2
9:54:33:4b
0:c:29:54:33:4b 0:c:29:1c:73:9f 0806 42: arp reply 10.10.10.2 is-at 0:c:2
9:54:33:4b
0:c:29:54:33:4b 0:c:29:1c:73:9f 0806 42: arp reply 10.10.10.2 is-at 0:c:2
9:54:33:4b
0:c:29:54:33:4b 0:c:29:1c:73:9f 0806 42: arp reply 10.10.10.2 is-at 0:c:2
9:54:33:4b
```

Figure 71: Shows the arp poisoning of the TACACS+ server and TACACS+ router. There is also another instance of this attack in another terminal, not shown (swapping the ip address order).

To verify that the ARP poisoning was successful, we went back to the console of the actual router, and issued a **show arp** command, to see all of the ARP information of devices connected to the router. See figure 72 below.

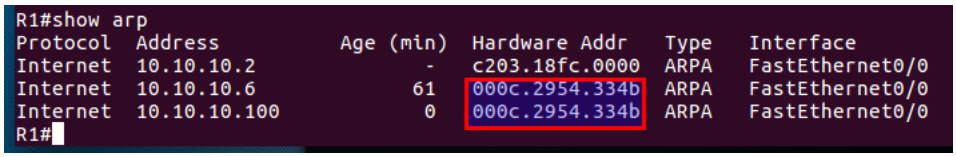


Figure 72: Proof that the arpspoof command was successful. In this figure, you can see that the attacker (10.10.10.6) has the same hardware or MAC address as the TACACS+ server (10.10.10.100). This shared address is `000c.2954.334b`.

Now that the ARP poisoning was successful, we ran a few `sysctl` and `iptables` commands in figure 73 so that the tacoflip.py script would be successful.

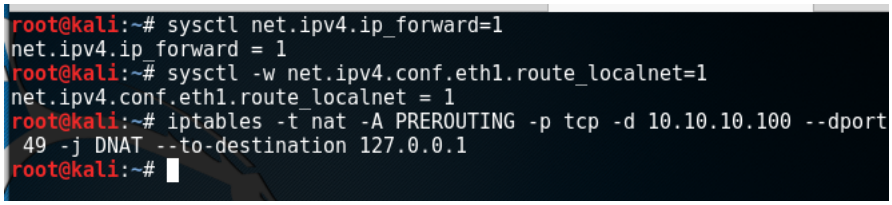


Figure 73: Shows the sysctl and iptables commands.

Q6: What do the first two "sysctl" commands do? What is the "iptables" command telling the TACACS+ server to do?
A6: The first sysctl command enables IP forwarding. This allows Kali to act as a "router" by being able to transfer packets from one IP address to another. The second sysctl command relates to the iptables command. By default, the iptables PREROUTING command does not work on the local loopback address in kali. This allows it to do so. The iptables command states that "all traffic destined for the TACACS+ server (10.10.10.100) should be routed to the local loopback address (127.0.0.01), on Kali".

After issuing these commands in figure 73, we now attempt to ssh into the router with invalid credentials in figure 74

Figure 74: Shows the attacker ssh-ing to the TACACS+ router with invalid credentials with the aid of the tacoflip.py file.

In figure 74, you can see that the ssh login required `-oKexAlgorithms=+diffie-hellman-group1-sha1` in order to be successful. This is due to the fact that our particular cisco router image offers an older version of key exchange when ssh is enabled (`diffie-hellman-group1-sha1`). This is a workaround to logging in on older versions of ssh.
Also in figure 74, you can see the successful login with invalid credentials to the TACACS+ router! We first have access to the "user exec" mode (>), then issue the `enable` command to access the "privilege exec" mode (#).
As soon as the attacker initiates the login command, the terminal running the tacoflip.py file shows information of the TACACS+ packets and proof of the bit flipping attack in figure 75.

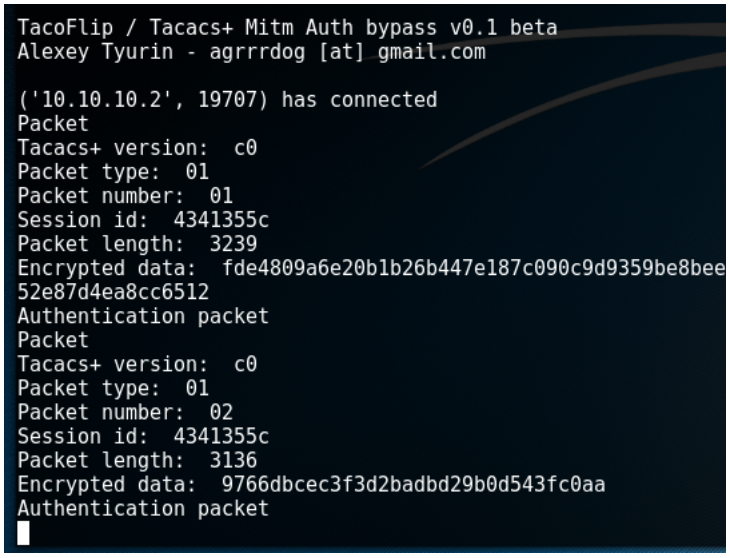


Figure 75: Shows the initial output of the tacoflip.py file when the attacker issues the first ssh command to login.

For further proof that we had full access to the router, we used commands such as `show ip int brief`, `show users`, and `show run` that we did in exercise 2. This can be seen below in figures 76, 77, and 78.

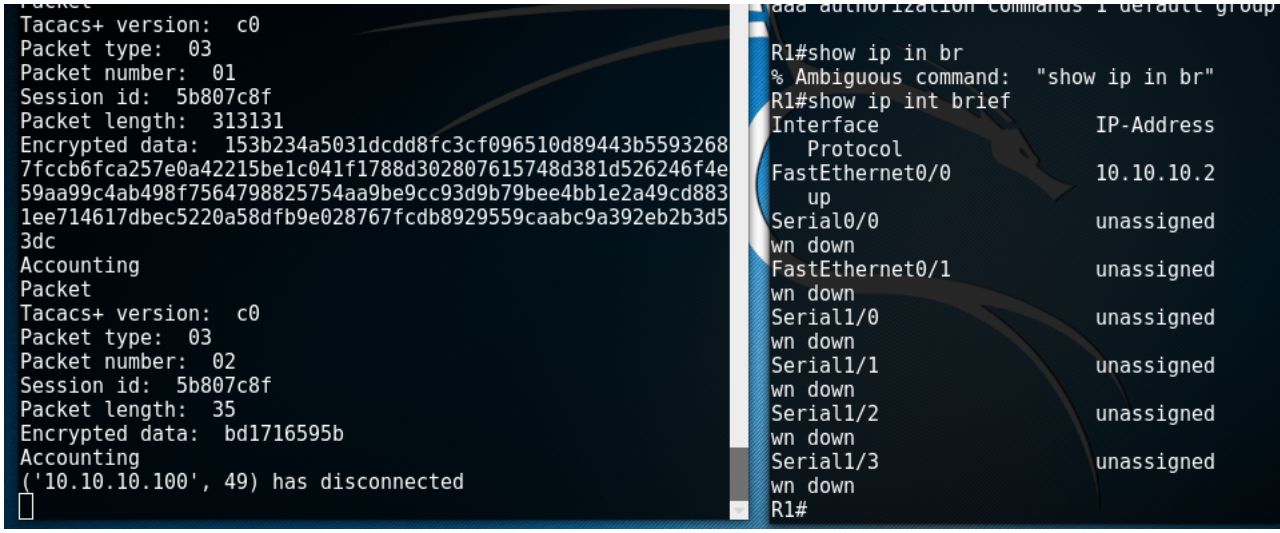


Figure 76: Issue of `show ip int brief` to see the available interfaces on the router, while logged in with the TacoTaco attack. Also in this figure, you can see the output of the tacoflip.py file

Figure 77: Issue of `show users` to see the currently logged in users while loggin in with the TacoTaco attack. Here you can see the user with invalid credentials logged in, and the address of where it is located.

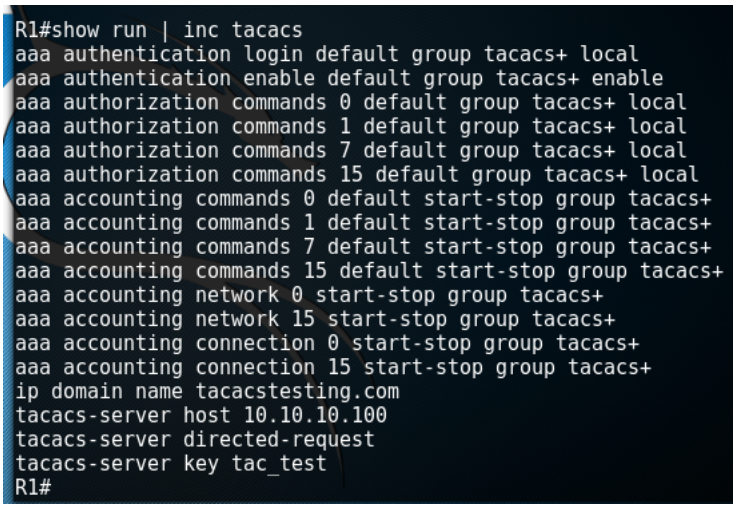


Figure 78: Issue of the `show run | inc tacacs` command while logged in with the TacoTaco attack. Here we use the "| inc tacacs" to only show the running configuration associated with the TACACS+ protocol.

We then exited the ssh shell, stopped wireshark, and exported the packets to a file called "tacotaco_packets.pcapng".
After issuing the command in figure 78, we are able to see the pre-shared-key that TACACS+ uses to encrypt the packet data. In our case, the key is "tac_test" (The one that we set in the configuration of the TACACS+ server in Exercise 1). Figure 79 below shows us giving the key to the wireshark capture to decrypt the encrypted requests of the entire TACACS+ packet body.

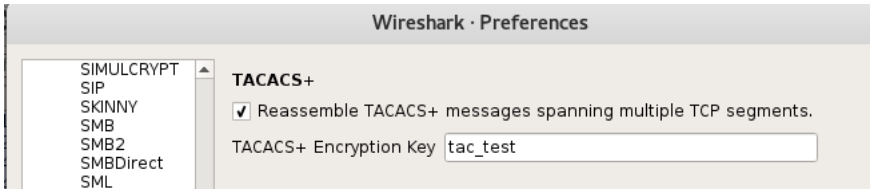


Figure 79: Edit - Preferences - Protocols - TACACS+

Once the key was given to wireshark, we are now able to see all aspects of the AAA conversation in plain text in the "decrypted request" field! This is shown in figures 80, 81, and 82.

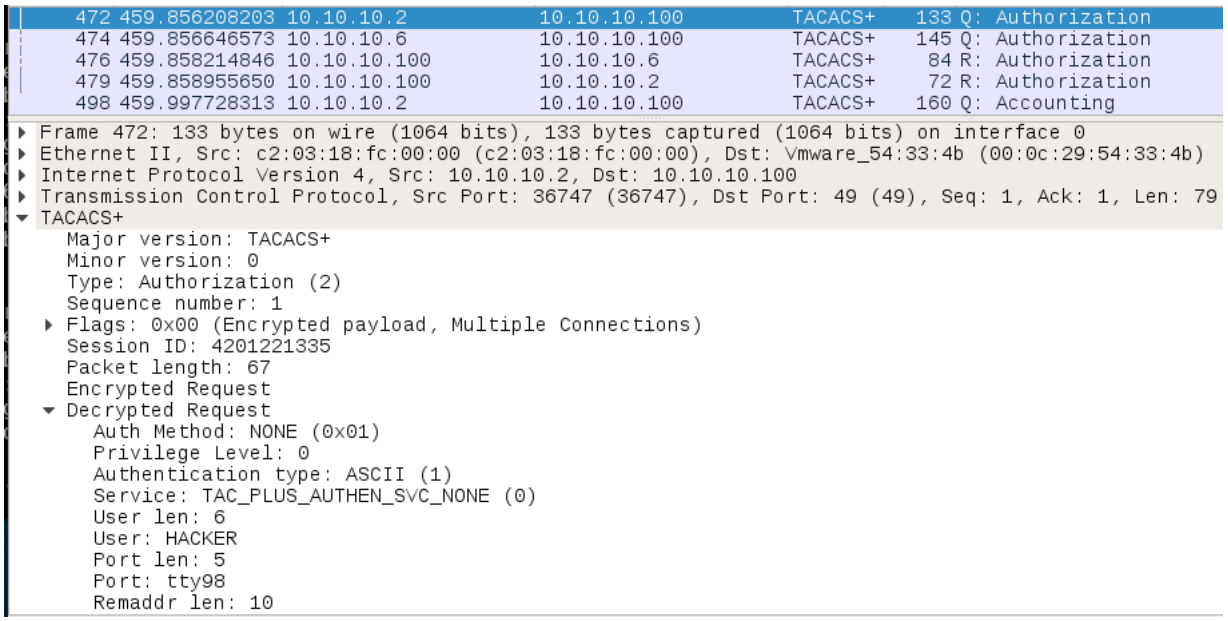


Figure 80: Decrypted Authorization packet. Here you can see the user name of the user who logged in, the privilege level that the user has, and more all in plain-text.

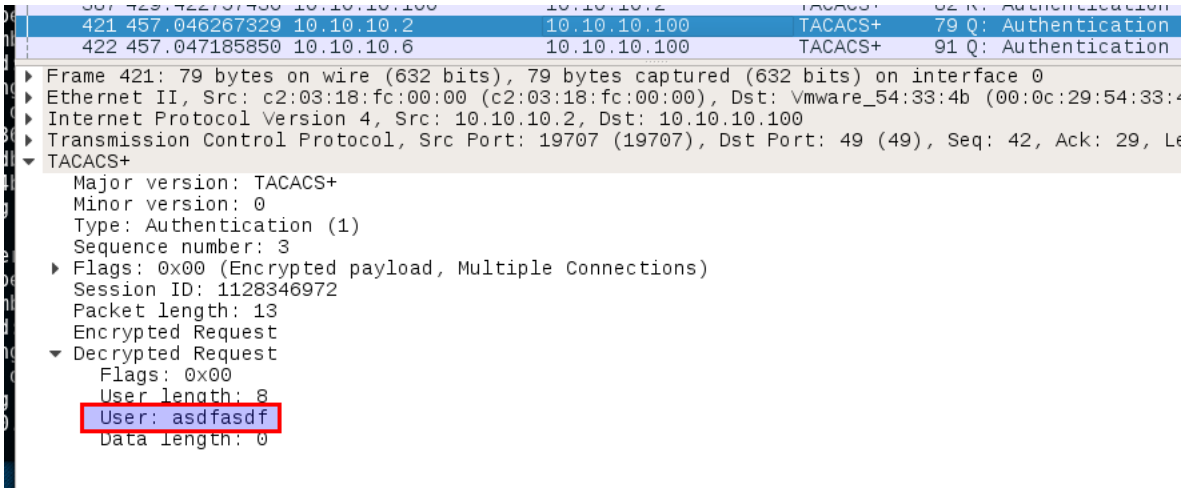


Figure 81: Decrypted Authentication packet. Here you can see the password that the user (attacker) inputted when the tacoflip.py attack was used.

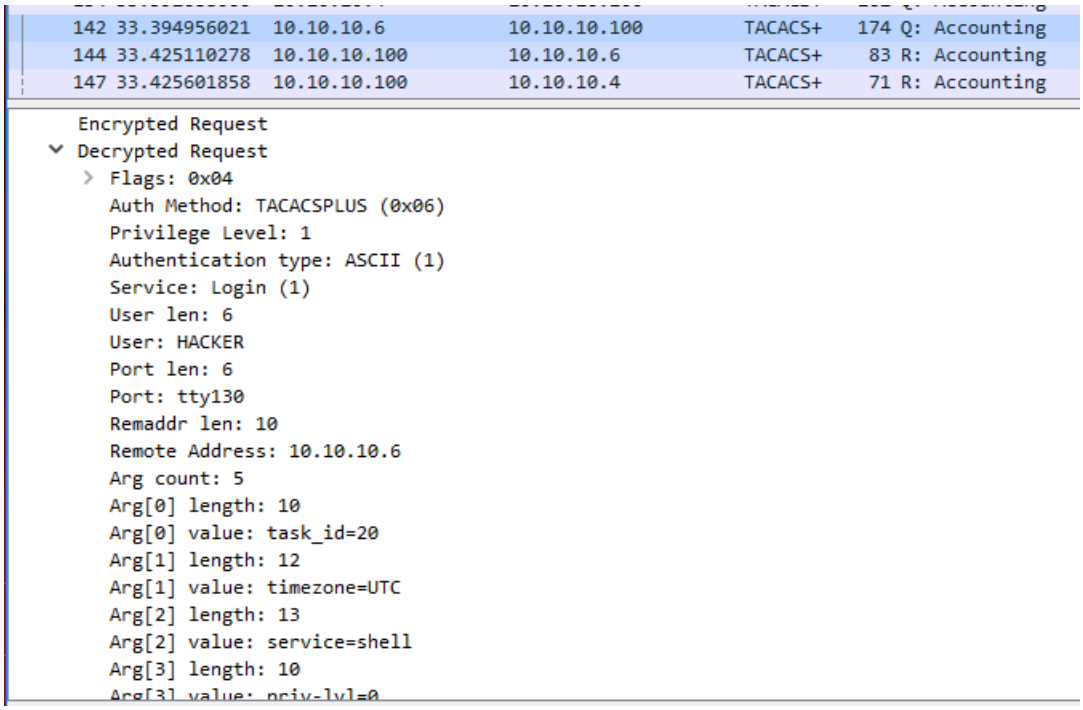


Figure 82: Decrypted Accounting packet. Here you can see auth method, username, remote address, the commands that the user inputted, and much more in plain-text.

After analyzing the wireshark capture, we went to the server vm to look at the log files after we completed the attack. These included the /var/log/-tac_plus.log and /var/log/tac_plus/tac_plus.acct. Screenshots of this can be found in the figures 83 and 84.

Figure 83: tac_plus.log

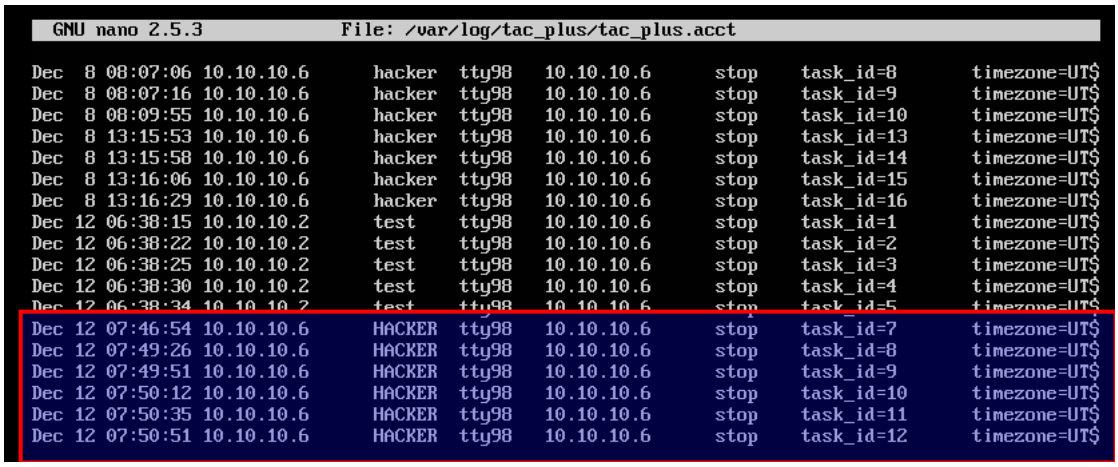


Figure 84: tac_plus.acct

From the tac_plus.log file, we saw that even though we had access from 10.10.10.6 as "HACKER", the log file counts the user as rejected. However, in the tac_plus.acct file, you can see what commands that the user inputted. This could be useful to a network administrator, so see if any sort of suspicious commands were used, or if any suspicious users were attempting to log in to the network device.

3.4.1.3 Step 3:Dynamic ARP Inspection / DHCP Snooping

In respect to time, we were not able to develop specific instructions or a lab report to implement and execute Dynamic Arp Inspection / DHCP Snooping. This is partially because the cisco router image we were provided with does not support the feature, as it is an outdated version. However, in the lab instructions, we will continue to describe what could happen if the feature was successfully implemented.

3.4.2 Key Learning/Takeaways:

If an attacker were to take a wireshark capture (through means of man-in-the-middle) of a legitimate login, that attacker would have the ability to perform the tacoflip.py attack and obtain the tacacs+ key. The attacker could then potentially login to the router to perform malicious activity on the router under the impersonation of a real user. Key takeaways from this exercise involved learning how TACACS+ encrypts the full body of its packets through its own algorithm, and examining the tacoflip.py file to learn exactly what was going on in the attack and how we were able to bypass all means of AAA. While it is possible to protect a network from Man-in-the-middle attacks, it appears that this protocol has an inherent vulnerability that will not be assessed, as the protocol is relatively old.

4 Lab Additional Questions & List of Attachments

4.1 Additional Questions

N/A

4.2 List of Attachments

- Exercise 1
  - Video Demo: N/A
- Exercise 2
  - Video Demo: https://www.youtube.com/watch?v=kE6lZISLnJQ
  - R_to_S_icmp.pcapng
  - S_to_R_icmp.pcapng
  - tacacs_telnet_1.pcapng
  - tacacs_test.pcapng
- Exercise 3
  - Video demo https://www.youtube.com/watch?v=I4Z1w40KTZ8
  - tacacs_dos.py
  - tacacs_scapy.py
  - tacacs_dos.pcapng
- Exercise 4
  - Video Demo: https://www.youtube.com/watch?v=SuQVGCM7kfA
  - TacoTaco-master.zip (contains tacoflip.py)
  - tacotaco_packets.pcapng

Go back to the introduction here: 1

5 Lab Observations, Suggestions & Best Practices

5.1 Observations

We discovered that the TACACS+ protocol is relatively old (no updated RFC since around 1998), but still relevant today. Because of this, it would still be possible to execute the attacks that we have in this lab in a fairly large infrastructure with no means of mitigation. While writing the lab report and lab instructions, Exercise 1 and 3 (Isaac) contain machines of different subnets / networks as 2 and 4 (Troy). This is because we were working in two different environments to finalize the project. Overall, we have learned much information in researching this protocol and will continue to utilize our skills in future "real-world" scenarios.

5.2 Suggestions

If we had more time to work on this project, we would continue to search for ways to successfully implement the DOS attack on the TACACS+ server, as well as the mitigation on MITM attacks.

5.3 Best Practices

Best practices included:

- Setting up a weekly meeting time to work on the project
- Organizing the lab exercises so that they flow nicely
- Contacting the creator of the TacoTaco project for assistance with Exercise 4 (Alexey Greendog Tyurin)
- Working together on the presentation and one-pager
- Emulating an environment in the lab on our laptops
- Creation of video demos

6 Lab References

[1] *Tacacs+ rfc.* [Online]. Available: https://tools.ietf.org/html/draft-grant-tacacs-02 (visited on 12/12/2016).

[2] *Greendog's blog attack on tacacs+ protocol.* [Online]. Available: http://agrrrdog.blogspot.com/2015/11/3-attacks-on-cisco-tacacs-bypassing.html (visited on 12/12/2016).

[3] *Arp spoof / poisoning explanation.* [Online]. Available: https://www.veracode.com/security/arp-spoofing (visited on 12/12/2016).

7 Acknowledgements

- We would like to acknowledge our Instructor, Dr. Emil Salib, for providing us with guidance in selecting our topic for the semester project, helping us structure our exercises in the correct order, and providing us with a cisco router image.
- We would also like to thank Alexey Greendog Tyurin, for his assistance with the TacoTaco attack.