Lab 3.1.3.4 – Linux Servers

Introduction

In this lab, you will use the Linux command line to identify servers running on a given computer.

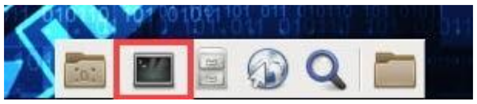
Part 1: Servers

Servers are essentially programs written to provide specific information upon request. Clients, which are also programs, reach out to the server, place the request and wait for the server response. Many different client-server communication technologies can be used, with the most common being IP networks. This lab focuses on IP network-based servers and clients.

Step 1: Access the command line.

a. Log on to the **CyberOps Workstation** VM as the `analyst`, using the password `cyberops`. The account *analyst* is used as the example user account throughout this lab.

b. To access the command line, click the **terminal** icon located in the *Dock*, at the bottom of VM screen. The terminal emulator opens.



Step 2: Display the services currently running.

Many different programs can be running on a given computer, especially a computer running a Linux operating system. Many programs run in the background so users may not immediately detect what programs are running on a given computer. In Linux, running programs are also called *processes*.

Note: The output of your *ps* command will differ because it will be based on the state of your *CyberOps Workstation* VM.

a. Use the `ps` command to display all the programs running in the background:

```
[analyst@secOps ~]$ sudo ps -elf
[sudo] password for analyst:
F S UID        PID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S root         1     0  0  80   0 -  2250 SyS_ep Feb27 ?        00:00:00 /sbin/init
1 S root         2     0  0  80   0 -     0 kthrea Feb27 ?        00:00:00 [kthreadd]
1 S root         3     2  0  80   0 -     0 smpboo Feb27 ?        00:00:00
[ksoftirqd/0]
```

```
1 S root         5     2  0  60 -20 -     0 worker Feb27 ?        00:00:00
[kworker/0:0H]
1 S root         7     2  0  80   0 -     0 rcu_gp Feb27 ?        00:00:00
[rcu_preempt]
1 S root         8     2  0  80   0 -     0 rcu_gp Feb27 ?        00:00:00 [rcu_sched]
1 S root         9     2  0  80   0 -     0 rcu_gp Feb27 ?        00:00:00 [rcu_bh]
1 S root        10     2  0 -40   - -     0 smpboo Feb27 ?        00:00:00
[migration/0]
1 S root        11     2  0  60 -20 -     0 rescue Feb27 ?        00:00:00 [lru-add-
drain]
5 S root        12     2  0 -40   - -     0 smpboo Feb27 ?        00:00:00
[watchdog/0]
1 S root        13     2  0  80   0 -     0 smpboo Feb27 ?        00:00:00 [cpuhp/0]
5 S root        14     2  0  80   0 -     0 devtmp Feb27 ?        00:00:00 [kdevtmpfs]
1 S root        15     2  0  60 -20 -     0 rescue Feb27 ?        00:00:00 [netns]
1 S root        16     2  0  80   0 -     0 watchd Feb27 ?        00:00:00
[khungtaskd]
1 S root        17     2  0  80   0 -     0 oom_re Feb27 ?        00:00:00
[oom_reaper]
<some output omitted
```

Why was it necessary to run *ps* as root (prefacing the command with *sudo*)?

When we start running the ps. Some may not display the output because some processes doesn't belongs to analyst.

b. In Linux, programs can also call other programs. The *ps* command can also be used to display such process hierarchy. Use `–ejH` options to display the currently running process tree.

Note: The process information for the *nginx* service is highlighted.

Note: If *nginx* is not running, enter the **sudo /usr/sbin/nginx** command at the command prompt to start the *nginx* service.

```
[analyst@secOps ~]$ sudo ps –ejH
[sudo] password for analyst:
<some output omitted>
  1     1     1 ?        00:00:00 systemd
  167   167   167 ?        00:00:01   systemd-journal
  193   193   193 ?        00:00:00   systemd-udevd
  209   209   209 ?        00:00:00   rsyslogd
  210   210   210 ?        00:01:41   java
  212   212   212 ?        00:00:01   ovsdb-server
  213   213   213 ?        00:00:00   start_pox.sh
  224   213   213 ?        00:01:18     python2.7
  214   214   214 ?        00:00:00   systemd-logind
  216   216   216 ?        00:00:01   dbus-daemon
  221   221   221 ?        00:00:05   filebeat
  239   239   239 ?        00:00:05   VBoxService
  287   287   287 ?        00:00:00   ovs-vswitchd
  382   382   382 ?        00:00:00   dhcpcd
  387   387   387 ?        00:00:00   lightdm
  410   410   410 tty7      00:00:10     Xorg
  460   387   387 ?         00:00:00     lightdm
  492   492   492 ?         00:00:00       sh
  503   492   492 ?         00:00:00         xfce4-session
  513   492   492 ?         00:00:00           xfwm4
  517   492   492 ?         00:00:00           Thunar
 1592   492   492 ?         00:00:00             thunar-volman
  519   492   492 ?         00:00:00           xfce4-panel
  554   492   492 ?         00:00:00             panel-6-systray
  559   492   492 ?         00:00:00             panel-2-actions
  523   492   492 ?         00:00:01           xfdesktop
  530   492   492 ?         00:00:00           polkit-gnome-au
  395   395   395 ?         00:00:00   nginx
  396   395   395 ?         00:00:00     nginx
  408   384   384 ?         00:01:58   java
  414   414   414 ?         00:00:00   accounts-daemon
  418   418   418 ?         00:00:00   polkitd
<some output omitted>
```

How is the process hierarchy represented by *ps*?

The process of hierarchy is represented by ps is using the independent.

c. As mentioned before, servers are essentially programs, often started by the system itself at boot time. The task performed by a server is called *service.* In such fashion, a web server provides web services.

The *netstat* command is a great tool to help identify the network servers running on a computer. The power of *netstat* lies on its ability to display network connections.

Note: Your output maybe different depending on the number of open network connections on your VM.

In the terminal window, type `netstat`.

```
[analyst@secOps ~]$ netstat
Active    Internet    connections    (w/o
servers)
Proto  Recv-Q  Send-Q  Local  Address                    Foreign  Address
State
tcp        0      0 localhost.localdo:48746 localhost.local:wap-wsp ESTABLISHED
tcp        0      0 localhost.localdo:48748 localhost.local:wap-wsp ESTABLISHED
tcp6       0      0 localhost.local:wap-wsp localhost.localdo:48748 ESTABLISHED
tcp6       0      0 localhost.local:wap-wsp localhost.localdo:48746 ESTABLISHED
tcp6       0      0 localhost.local:wap-wsp localhost.localdo:48744 ESTABLISHED
tcp6       0      0 localhost.localdo:48744 localhost.local:wap-wsp ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto  RefCnt Flags          Type          State            I-Node
Path
unix  3      [ ]         DGRAM                    8472     /run/systemd/notify
unix  2      [ ]         DGRAM                    8474     /run/systemd/cgroups-
agent<some          output
omitted>
```

As seen above, *netstat* returns lots of information when used without options. Many options can be used to filter and format the output of *netstat*, making it more useful.

d. Use **netstat** with the **–tunap** options to adjust the output of *netstat*. Notice that *netstat* allows multiple options to be grouped together under the same “**-**“ sign.

The information for the *nginx* server is highlighted.

```
[analyst@secOps ~]$ sudo netstat -tunap
[sudo] password for analyst:
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
395/nginx: master p
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN
279/vsftpd
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
277/sshd
tcp        0      0 0.0.0.0:6633            0.0.0.0:*               LISTEN
257/python2.7
tcp6       0      0 :::22                   :::*                    LISTEN
277/sshd
tcp6       0      0 :::23                   :::*                    LISTEN
1/init
```

What is the meaning of the *–t*, *-u*, *–n*, *–a* and *–p* options in *netstat*? (use *man netstat* to answer)

- -t represents Transmission Control Protocol connection.
- -u represents UDP connection.
- -n represents numeric output.
- -a represents both Listen & Non-Listen.
- -p represents PID connection.

Is the order of the options important to *netstat*?
No

Clients will connect to a port and, using the correct protocol, request information from a server. The *netstat* output above displays a number of services that are currently listening on specific ports. Interesting columns are:

- The first column shows the Layer 4 protocol in use (UDP or TCP, in this case).
- The third column uses the *<ADDRESS:PORT>* format to display the local IP address and port on which a specific server is reachable. The IP address 0.0.0.0 signifies that the server is currently listening on all IP addresses configured in the computer.
- The fourth column uses the same socket format *<ADDRESS:PORT>* to display the address and port of the device on the remote end of the connection. 0.0.0.0:* means that no remote device is currently utilizing the connection.
- The fifth column displays the state of the connection.
- The sixth column displays the process ID (*PID*) of the process responsible for the connection. It also displays a short name associated to the process.

Based on the *netstat* output shown in item (d), what is the Layer 4 protocol, connection status, and PID of the process running on port 80?

TCP, LISTEN and 391

While port numbers are just a convention, can you guess what kind of service is running on port 80 TCP?

Web Server

e. Sometimes it is useful to cross the information provided by *netstat* with *ps*. Based on the output of item (d), it is known that a process with *PID 395* is bound to TCP port 80. Port 395 is used in this example. Use **ps** and **grep** to list all lines of the *ps* output that contain **PID 395**.

Note: Replace the *PID* with the *PID* shown on your output as they may differ from this example.

```
[analyst@secOps ~]$ sudo ps -elf | grep 395
[sudo] password for analyst:
1 S root       395     1  0  80   0 -  1829 sigsus Feb27 ?        00:00:00 nginx:
master process /usr/bin/nginx -g pid /run/nginx.pid; error_log stderr;
5 S http       396   395  0  80   0 -  1866 SyS_ep Feb27 ?        00:00:00 nginx:
worker process
0 S analyst   3789  1872  0  80   0 -  1190 pipe_w 14:05 pts/1    00:00:00 grep 395
```

In the output above, the *ps* command is *piped* through the *grep* command to filter out only the lines containing the number 395. The result is three lines with text wrapping.

The first line shows a process owned by the *root* user (third column), started by another process with PID *1* (fifth column), on *Feb27* (twelfth column) with command */usr/bin/nginx -g pid /run/nginx.pid; error_log stderr;*

The second line shows a process with *PID 396*, owned by the *http* user, started by process *395*, on *Feb27*.

The third line shows a process owned by the *analyst* user, with *PID 3789*, started by a process with PID 1872, as the *grep 395* command.

The process *PID 395* is *nginx*. How could that be concluded from the output above?

Output is nginx command line.

What is *nginx*? What is its function? (Use *google* to learn about *nginx*)

Nginx is a software. It is mostly used for proxy purposes. It also notices the syntax error in files. It also supported in java, Node.js and Ruby. where it has many other features like configuration, balancing and routing.

The second line shows that process *396* is owned by a user named *http* and has process number *395* as its parent process. What does that mean? Is this common behavior?

The nginx started the process under the name of http username and it always connect the port of 80 to TCP.

Why is the last line showing *grep 395*?

When the ouput were complied, then grep 365 was still running and it helps to ps output.

Part 2: Using Telnet to Test TCP Services

Telnet is a simple remote shell application. *Telnet* is considered insecure because it does not provide encryption. Administrators who choose to use *Telnet* to remotely manage network devices and servers will expose login credentials to that server, as Telnet will transmit session data in clear text. While Telnet is not recommended as a remote shell application, it can be very useful for quickly testing or gathering information about TCP services.

The Telnet protocol operates on port 23 using TCP by default. The *telnet* client however, allows for a different port to be specified. By changing the port and connecting to a server, the *telnet* client allows for a network analyst to quickly assess the nature of a specific server by communicating directly to it.

Note: It is strongly recommended that *ssh* be used as remote shell application instead of *telnet*.

a. In Part 1, *nginx* was found to be running and assigned to port 80 TCP. Although a quick *Google* search revealed that *nginx* is a lightweight web server, how would an analyst be sure of that? What if an attacker changed the name of a malware program to *nginx*, just to make it look like the popular webserver? Use **telnet** to connect to the local host on port 80 TCP:

```
[analyst@secOps ~]$ telnet 127.0.0.1 80
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is
'^]'.
```

b. Press a few letters on the keyboard. Any key will work. After a few keys are pressed, press **ENTER**. Below is the full output, including the Telnet connection establishment and the random keys pressed (*fdsafsdaf*, this case):

```
fdsafsdaf
HTTP/1.1 400 Bad Request
Server: nginx/1.12.0
Date: Tue, 28 Feb 2017 20:09:37
GMT Content-Type: text/html
Content-Length: 173
Connection: close

<html>
<head><title>400 Bad Request</title></head>
<body bgcolor="white">
<center><h1>400 Bad Request</h1></center>
<hr><center>nginx/1.10.2</center>
</body>
</html>
Connection closed by foreign host.
```

Thanks to the *Telnet* protocol, a clear text *TCP* connection was established, by the *Telnet* client, directly to the *nginx* server, listening on *127.0.0.1 port 80 TCP*. This connection allows us to send data directly to the server. Because *nginx* is a web server, it does not understand the sequence of random letters sent to it and returns an error in the format of a web page.

Why was the error sent as a web page?

Nginx is a Webserver. So, that is the main reason it sent an error.

While the server reported an error and terminated the connection, we were able to learn a lot. We learned that:

1) The *nginx* with *PID 395* is in fact a web server.
2) The version of *nginx* shown in the lab output is *1.12.0*.
3) The network stack of our *CyberOps Workstation* VM is fully functional all the way to *Layer 7*.

Not all services are equal. Some services are designed to accept unformatted data and will not terminate if garbage is entered via keyboard. Below is an example of such a service:

c. Looking at the *netstat* output presented earlier, it is possible to see a process attached to port 22. Use **Telnet** to connect to it.

Port 22 TCP is assigned to *SSH* service. *SSH* allows an administrator to connect to a remote computer securely.

Below is the output:

```
[analyst@secOps ~]$ telnet 127.0.0.1 22
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is
'^]'. SSH-2.0-
OpenSSH_7.4 sdfjlskj
Protocol mismatch.
Connection closed by foreign host.
```

Use **Telnet** to connect to **port 68**. What happens? Explain.

It shows Unable to connect the remote host. Telnet is a TCP port where has It will not be able to connect the UDP ports.

.

Reflection

What are the advantages of using netstat?

Netstat, it has information about the routing tables its supports cmp, it also helps in displaying kernel route information. Lists UDP and TCP ports for network running in system.

What are the advantages of using Telnet? Is it safe?

Yes, It is safe and secure to quick test of combining information about the different types of networks services.