

Comp 4108 – Computer Systems Security
Experiences 2

Student Name: Ben Cendana
Student #: 100811212

Table Of Contents

Contents

1.0: Introduction	2
2.0: The Replacement Passwd Program	2
2.1: Preliminary Linux Binaries.....	2
2.1.2: Preliminary Linux Binaries - The Chsh Binary.....	2
2.1.3 Preliminary Linux Binaries - The chfn Binary.....	3
2.2: The Passwd Program.....	4
2.2.1: Step 1 - Create A Binary Script	4
2.2.2: Step 2 – Compile The rootScript.c file.....	6
2.2.3: Step 3 – Test The Root Access	7
2.2.4: Step 4 – Getting Root Access	7
2.2.5: Step 5: Running The RootScripts.....	8
3.0: Setuid Root Binaries and Capabilities	9
3.1: Determining Capabilities.....	9
3.1.1: Determining Capabilities - Passwd.....	10
3.1.2: Determining Capabilities - Mount, Ping and Su	10
3.2: Remove Set UID Bit and Set Cap	10
4.0: Restricting Network Access.....	12
4.1: Blocking Network Traffic.....	13
4.1.1: Blocking Network Traffic -Protocols	13
4.1.2: Blocking Network Traffic -IP.....	13
4.1.3: Blocking Network Traffic -Application	13
4.2: Testing Network Traffic.....	15
4.3: Implementing A Partial Block.....	16

1.0: Introduction

This report documents the steps performed to accomplish experience #2 for Comp 4108 – Computer Security. The experience was divided into three parts each exploring a related theme these are:

1. The Replacement Passwd Program
2. Setuid Root Binaries and Capabilities
3. Restricting Network Access

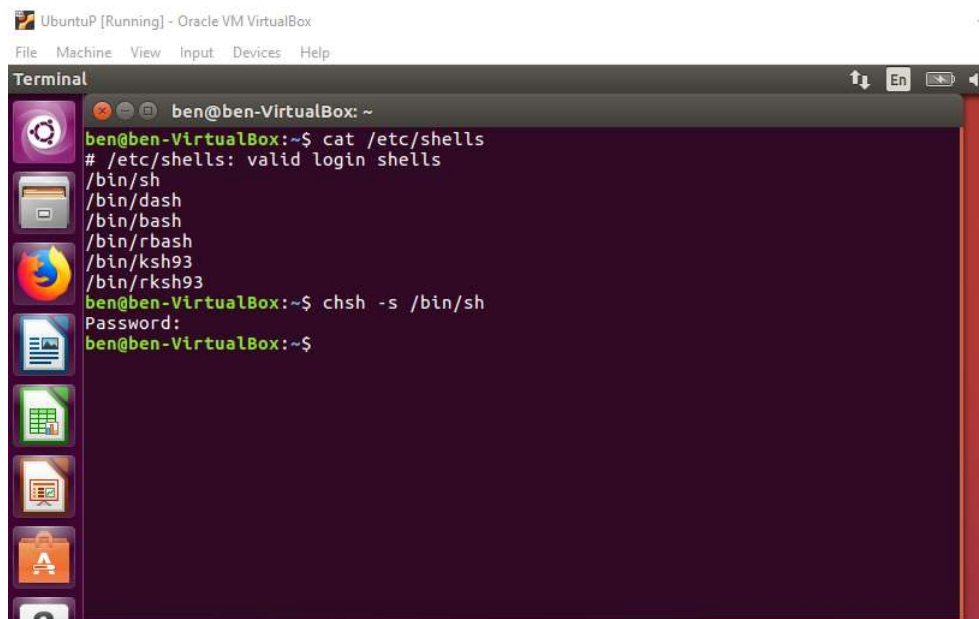
2.0: The Replacement Passwd Program

2:1: Preliminary Linux Binaries

The instructions for this module suggested that we should first try using the **chsh** and **passwd** commands as it maybe challenging to create a properly hashed password.

2.1.2: Preliminary Linux Binaries - The Chsh Binary

The **chsh** command is a Linux binary that changes the program or shell that processes the users input.

A screenshot of a terminal window titled "UbuntuP [Running] - Oracle VM VirtualBox". The terminal shows a user named "ben" at a prompt "ben@ben-VirtualBox: ~". The user enters the command "cat /etc/shells", which outputs a list of valid login shells: "/bin/sh", "/bin/dash", "/bin/bash", "/bin/rbash", "/bin/ksh93", and "/bin/rksh93". The user then enters the command "chsh -s /bin/sh", which prompts for a "Password:". The terminal window has a sidebar with various application icons and a top menu bar with "File", "Machine", "View", "Input", "Devices", and "Help".

```
ben@ben-VirtualBox: ~  
ben@ben-VirtualBox:~$ cat /etc/shells  
# /etc/shells: valid login shells  
/bin/sh  
/bin/dash  
/bin/bash  
/bin/rbash  
/bin/ksh93  
/bin/rksh93  
ben@ben-VirtualBox:~$ chsh -s /bin/sh  
Password:  
ben@ben-VirtualBox:~$
```

Figure 1: The Chsh Binary Being Called

Before we change our current shell its a good idea to see what shells we have available, by typing in the command **cat /etc/shells** we can display all the available shells we can use. To change a shell we type the command **chsh -s /bin/sh** which changes the shell to /bin/sh.

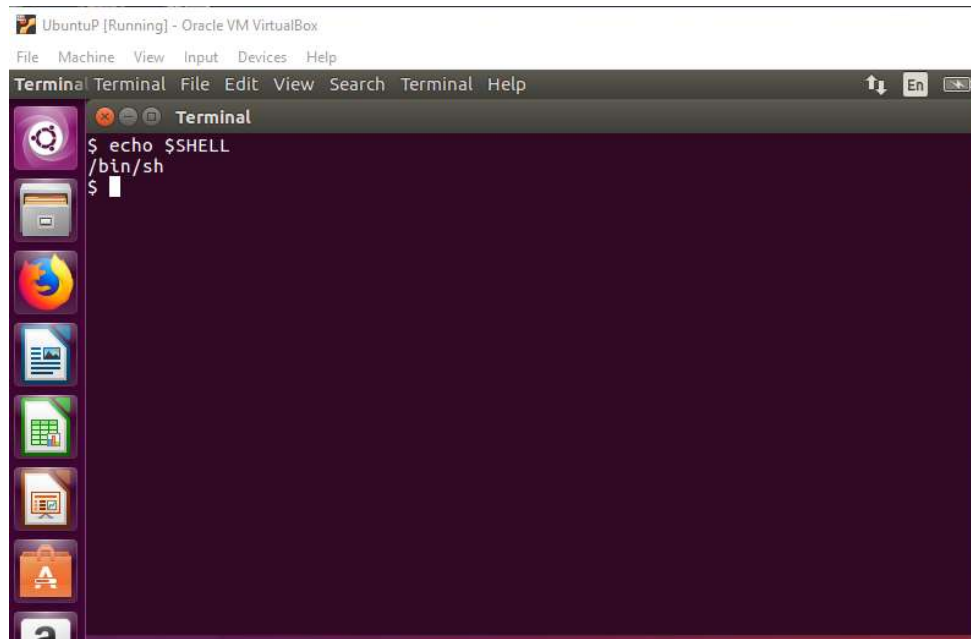


Figure 2: Checking To See If The Shell Has Been Changed

The last step is to verify if the shell was successfully changed by using the **echo \$SHELL** command we can verify that it was changed to `/bin/sh`.

2.1.3 Preliminary Linux Binaries - The chfn Binary

The change finger binary is a binary that allows users to change information about selected users.

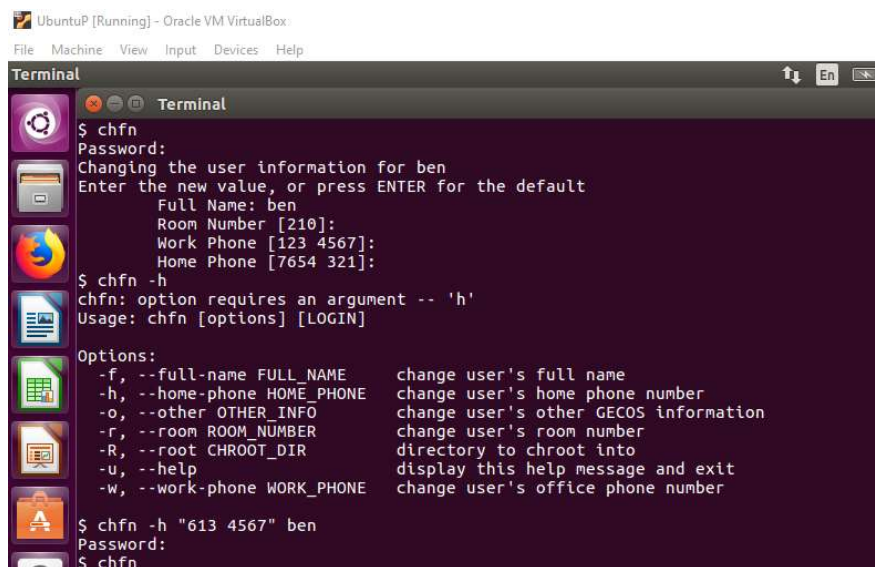
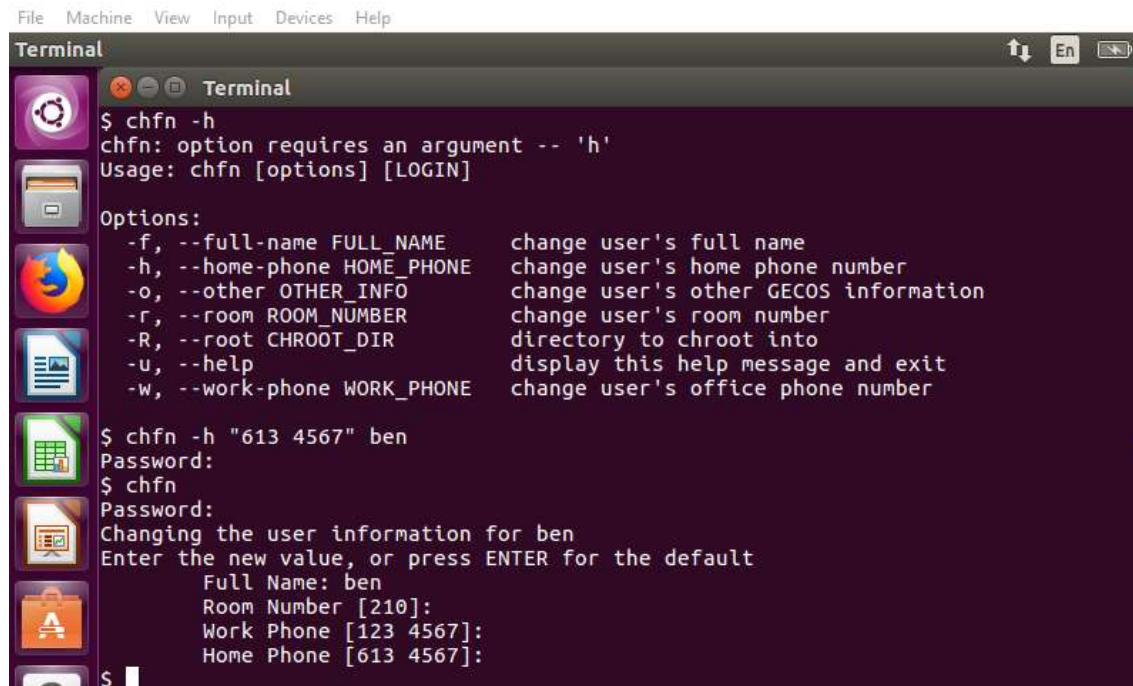


Figure 3: The chfn Binary

The example in Figure 3 shows the **chfn** command being called displaying the finger information for the current user and the **chfn -h** command being called displaying all the options we have for change finger.



```
File Machine View Input Devices Help
Terminal
$ chfn -h
chfn: option requires an argument -- 'h'
Usage: chfn [options] [LOGIN]

Options:
  -f, --full-name FULL_NAME    change user's full name
  -h, --home-phone HOME_PHONE  change user's home phone number
  -o, --other OTHER_INFO       change user's other GECOS information
  -r, --room ROOM_NUMBER       change user's room number
  -R, --root CHROOT_DIR        directory to chroot into
  -u, --help                   display this help message and exit
  -w, --work-phone WORK_PHONE  change user's office phone number

$ chfn -h "613 4567" ben
Password:
$ chfn
Password:
Changing the user information for ben
Enter the new value, or press ENTER for the default
Full Name: ben
Room Number [210]:
Work Phone [123 4567]:
Home Phone [613 4567]:
$
```

Figure 4: The chfn Binary

In Figure 4 we can see **chfn -h "613 4567" ben** command being typed which changes the current users phone number from 7654 321 to 613 4567.

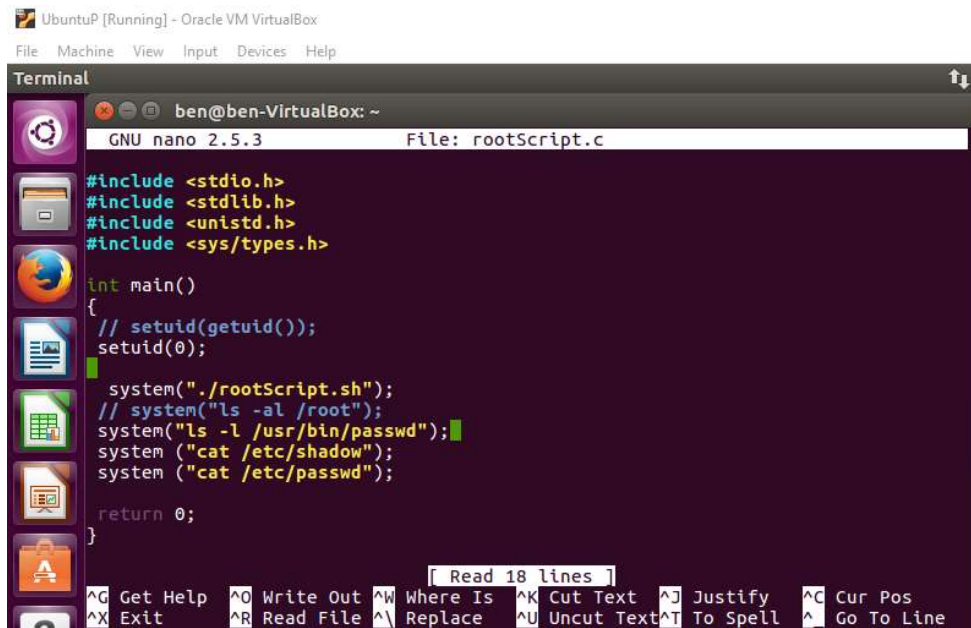
2.2: The Passwd Program

This program changes the password of the current user.

2.2.1: Step 1 - Create A Binary Script

In order for this to work we will create three scripts written in 3 different languages each called rootScript, all of which make system calls to root.

The first script was written in c, followed by a second script written in python and the last one written as a shell.



The screenshot shows a terminal window titled "UbuntuP [Running] - Oracle VM VirtualBox". The terminal is running the GNU nano 2.5.3 editor, editing the file "rootScript.c". The code is as follows:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

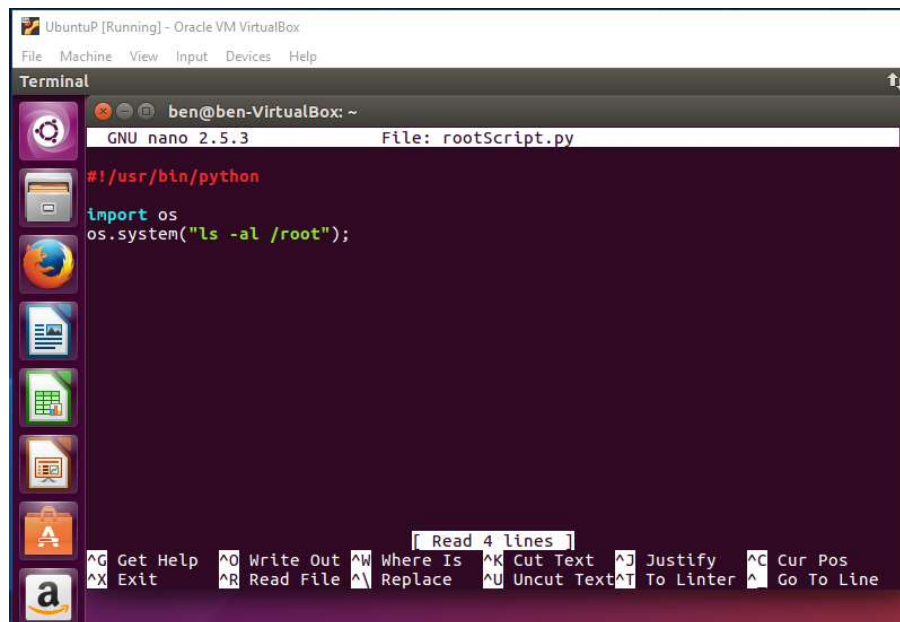
int main()
{
    // setuid(getuid());
    setuid(0);

    system("./rootScript.sh");
    // system("ls -al /root");
    system("ls -l /usr/bin/passwd");
    system("cat /etc/shadow");
    system("cat /etc/passwd");

    return 0;
}
```

The terminal window also shows a sidebar with various application icons and a bottom status bar with keyboard shortcuts.

Figure 5: The rootScript.c file



The screenshot shows a terminal window titled "UbuntuP [Running] - Oracle VM VirtualBox". The terminal is running the GNU nano 2.5.3 editor, editing the file "rootScript.py". The code is as follows:

```
#!/usr/bin/python

import os
os.system("ls -al /root");
```

The terminal window also shows a sidebar with various application icons and a bottom status bar with keyboard shortcuts.

Figure 6: The rootScript.py Script

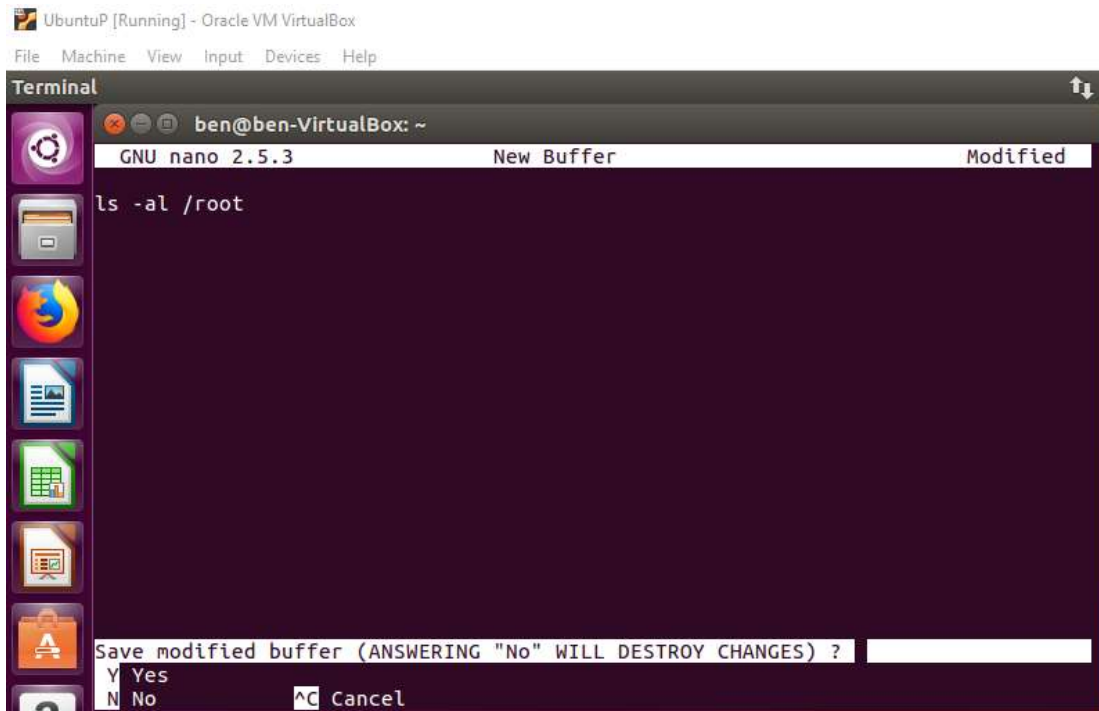


Figure 7: The rootScript.sh file

As we can see in the rootScript.c file we also make system call to display and change the user password, these commands are: **cat /etc/shadow** and **cat /etc/passwd**.

2.2.2: Step 2 – Compile The rootScript.c file

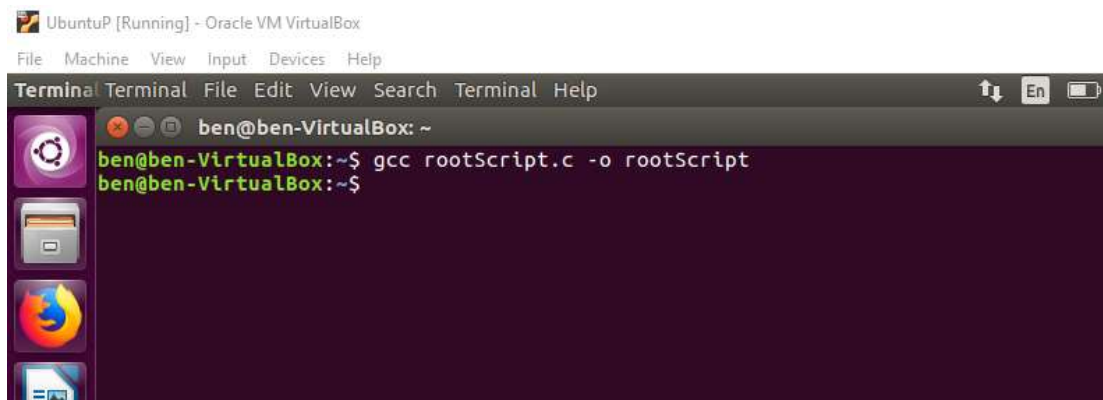


Figure 8: Compiling The rootScript.c file

As can be seen in Figure 8 the rootScript.c file is compiled using the **gcc rootScript.c -o rootScript** command.

2.2.3: Step 3 – Test The Root Access

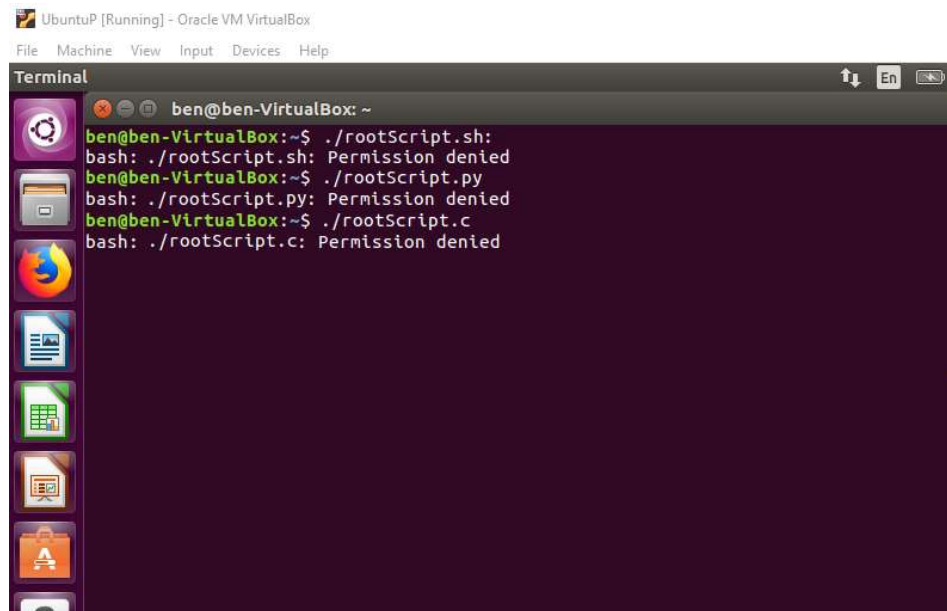


Figure 8: Testing for root access

Figure 5 shows that we are unable to run any of the scripts because they are not a member of the user group root.

2.2.4: Step 4 – Getting Root Access

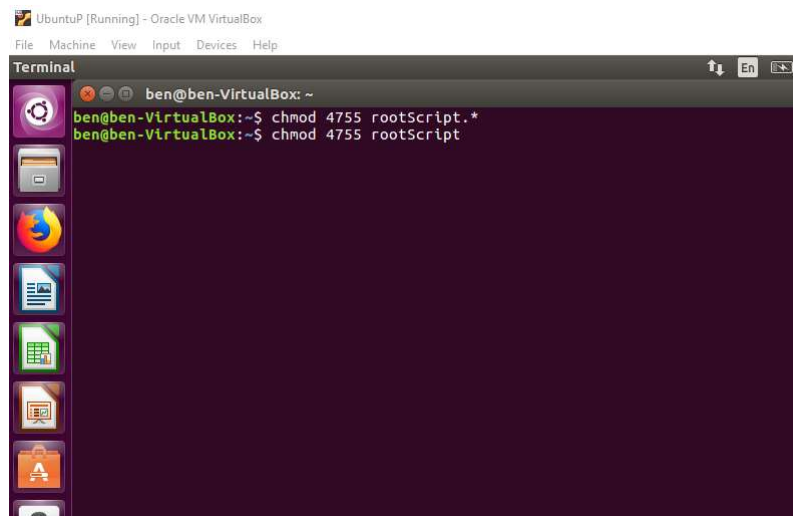
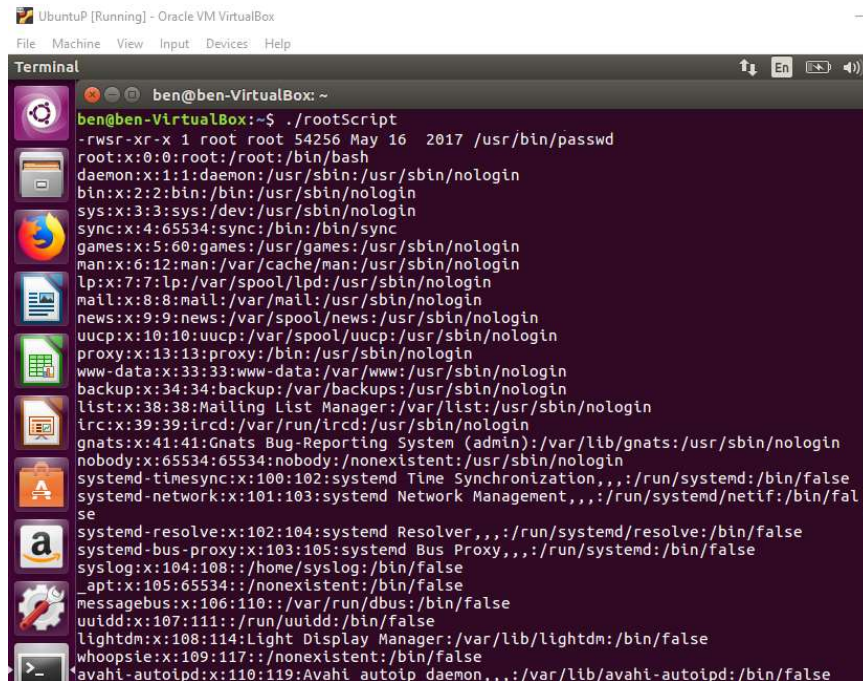


Figure 9: Using The chmod Command

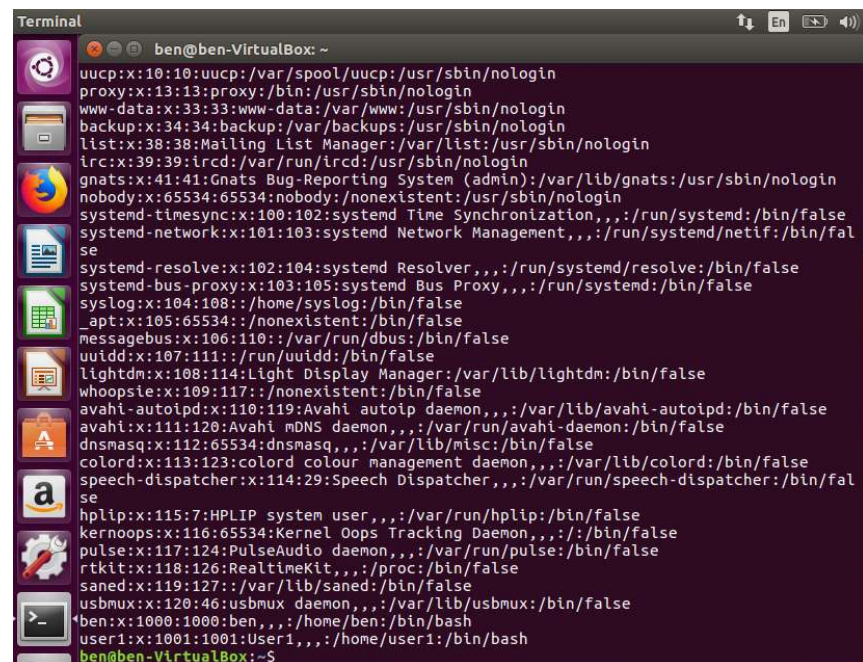
In order to get root capabilities we use the **chmod 4755** to change the access Permissions (Figure 9).

2.2.5: Step 5: Running The RootScripts



```
ben@ben-VirtualBox:~$ ./rootScript
-rwsr-xr-x 1 root root 54256 May 16 2017 /usr/bin/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:103:systemd Network Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false
syslog:x:104:108::/home/syslog:/bin/false
_apt:x:105:65534::/nonexistent:/bin/false
messagebus:x:106:110::/var/run/dbus:/bin/false
uidd:x:107:111::/run/uidd:/bin/false
lightdm:x:108:114:Light Display Manager:/var/lib/lightdm:/bin/false
whoopie:x:109:117::/nonexistent:/bin/false
avahi-autoipd:x:110:119:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false
```

Figure 10: Displaying The User Passwords



```
ben@ben-VirtualBox:~$ ./rootScript
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:103:systemd Network Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false
syslog:x:104:108::/home/syslog:/bin/false
_apt:x:105:65534::/nonexistent:/bin/false
messagebus:x:106:110::/var/run/dbus:/bin/false
uidd:x:107:111::/run/uidd:/bin/false
lightdm:x:108:114:Light Display Manager:/var/lib/lightdm:/bin/false
whoopie:x:109:117::/nonexistent:/bin/false
avahi-autoipd:x:110:119:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false
avahi:x:111:120:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/bin/false
colord:x:113:123:colord colour management daemon,,,:/var/lib/colord:/bin/false
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/false
hplip:x:115:7:HPLIP system user,,,:/var/run/hplip:/bin/false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/bin/false
pulse:x:117:124:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
ben:x:1000:1000:ben,,,:/home/ben:/bin/bash
user1:x:1001:1001:User1,,,:/home/user1:/bin/bash
ben@ben-VirtualBox:~$
```

Figure 11: Changing The User Passwords

As we can see in Figure 9 we can view all the user passwords, in Figure 10 we change the password for ben.

3.0: Setuid Root Binaries and Capabilities

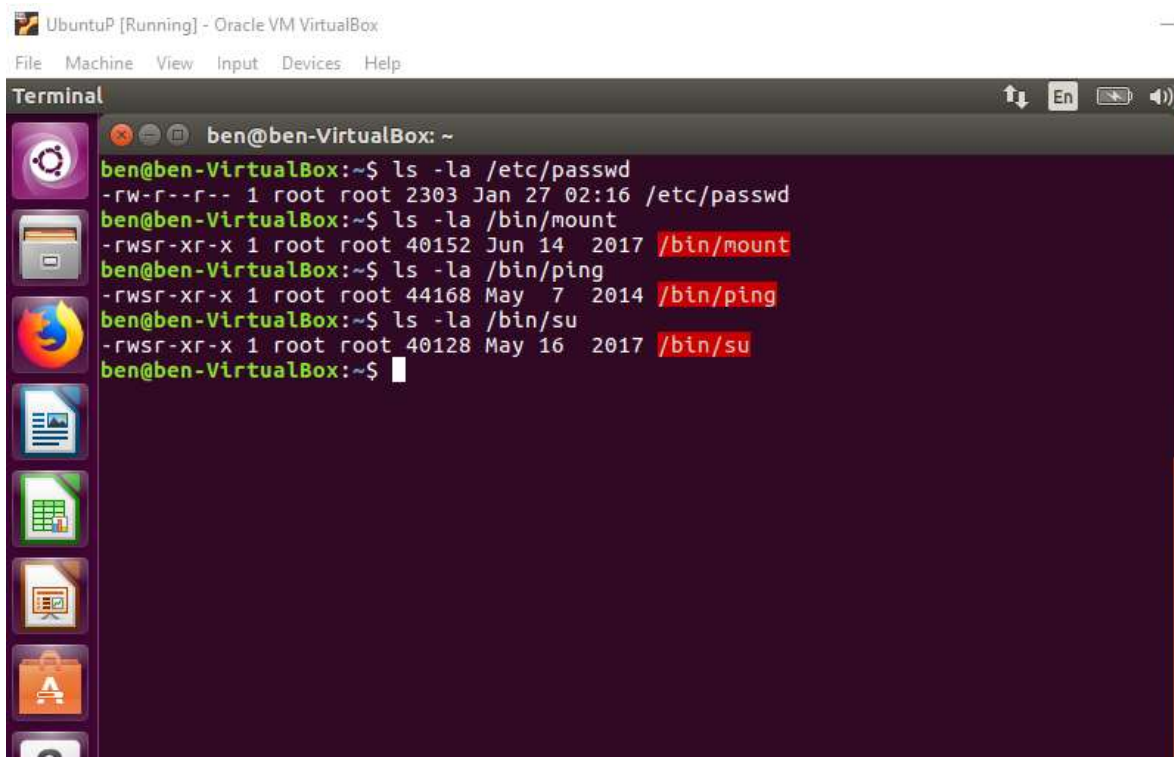
The word capabilities in a computer science realm means certain processes are allowed to run on an object. Within the confines of Linux this means that only certain processes can be run because certain binaries require higher level privileges in order to be run.

This module will explore four suggested setuid binaries these are: passwd, mount, ping and sudo.

3.1: Determining Capabilities

Processes and binaries can be grouped under two categories of permissions privileged and unprivileged. Whenever a process has a UID of 0 it has been granted privileged permission. If a process has a UID of anything but 0 we know that some processes will require permissions to run. When that occurs the UID and GID are checked to see if the process should be granted permissions to run or not.

We determine the capabilities of each processes/binaries without reading there documentation or source by using the **ls** command and having there UID and GID displayed.



```
ben@ben-VirtualBox: ~  
ben@ben-VirtualBox:~$ ls -la /etc/passwd  
-rw-r--r-- 1 root root 2303 Jan 27 02:16 /etc/passwd  
ben@ben-VirtualBox:~$ ls -la /bin/mount  
-rwsr-xr-x 1 root root 40152 Jun 14 2017 /bin/mount  
ben@ben-VirtualBox:~$ ls -la /bin/ping  
-rwsr-xr-x 1 root root 44168 May 7 2014 /bin/ping  
ben@ben-VirtualBox:~$ ls -la /bin/su  
-rwsr-xr-x 1 root root 40128 May 16 2017 /bin/su  
ben@ben-VirtualBox:~$
```

Figure 12: The setuid binaries

3.1.1: Determining Capabilities - Passwd

As can be seen in Figure 12 the passwd binary has a UID of 1, and has -rw-r -- r capabilities which means the file can be read and write. However only -r --r is available to both the user and user group root, thus the user can only be root to use this binary.

3.1.2: Determining Capabilities - Mount, Ping and Su

As can be seen in Figure 12 the mount, ping and su binaries each have a UID of 1 and each has -rwsr capabablites, what is interesting is the -s capability that all three binaries share.

The -s capability stands for setuid, it means that a non root user may gain temporary capabilities to use this process and his/her id will temporally become root to use this binary, once the process is completed all root privileges will then be lost.

3.2: Remove Set UID Bit and Set Cap

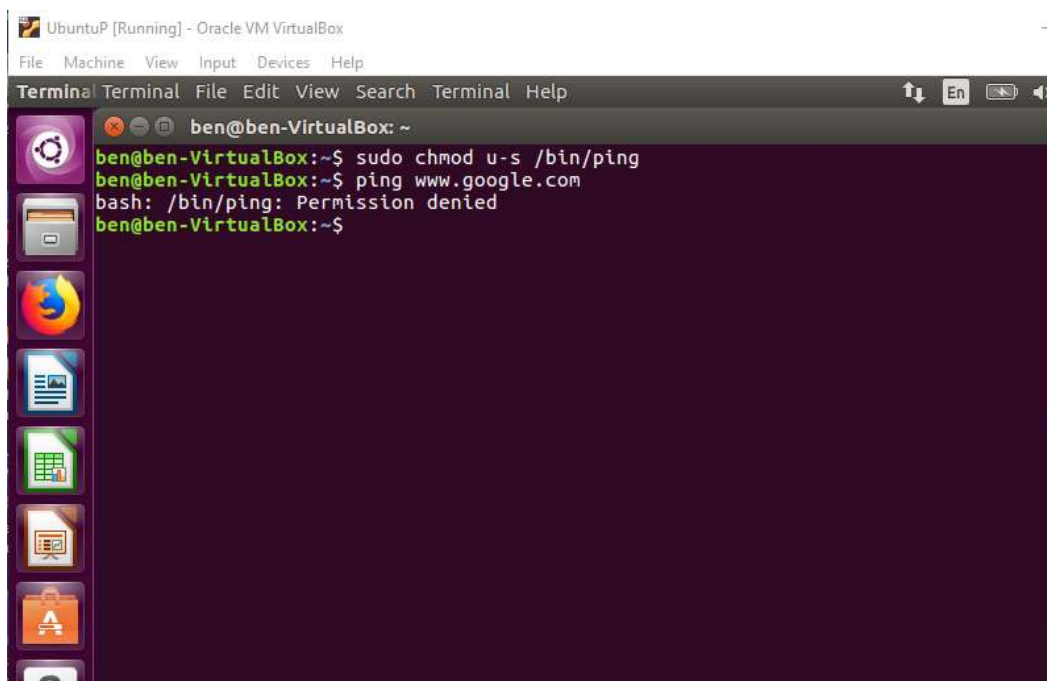


Figure 13: Using the SetUI To Limit Capacity

Removing capabilities can be done by using the command **chmod u-s /filepath**, this removes the set id. As can be seen in Figure 13 the set id was successfully removed for the ping binary.

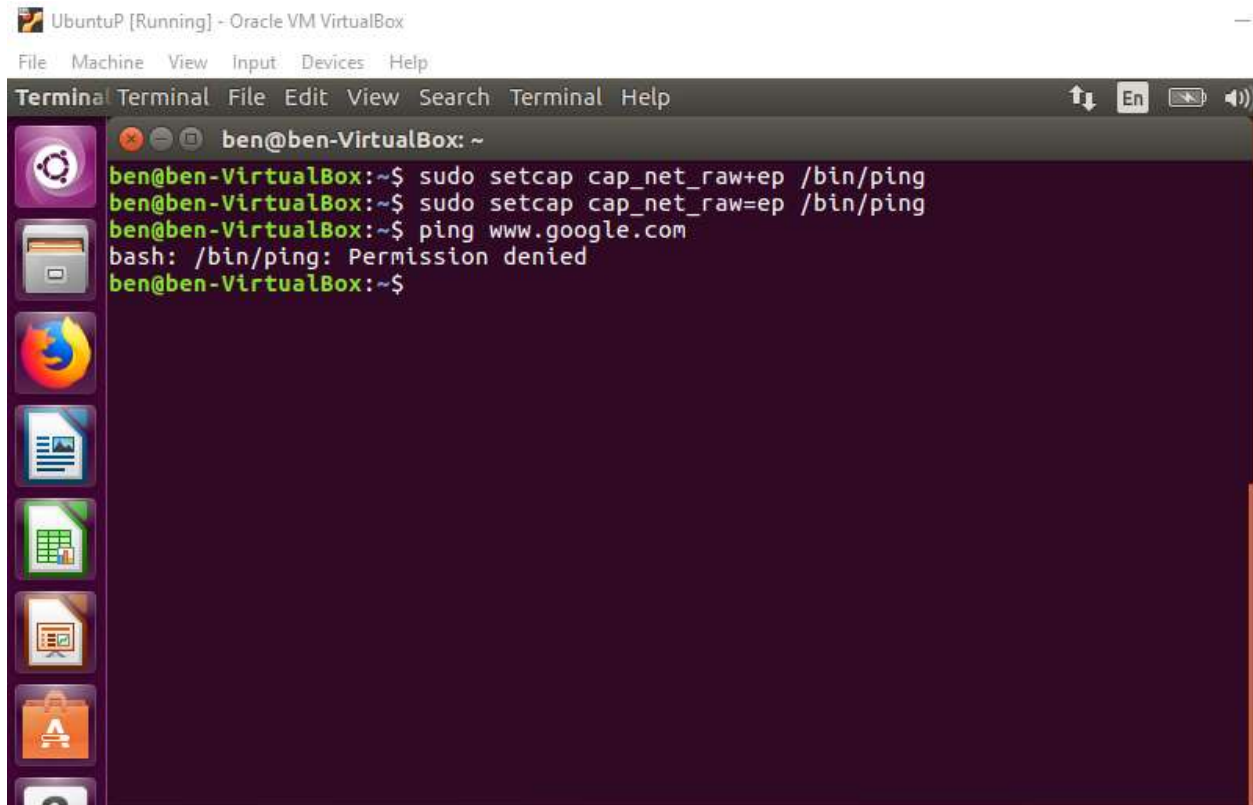
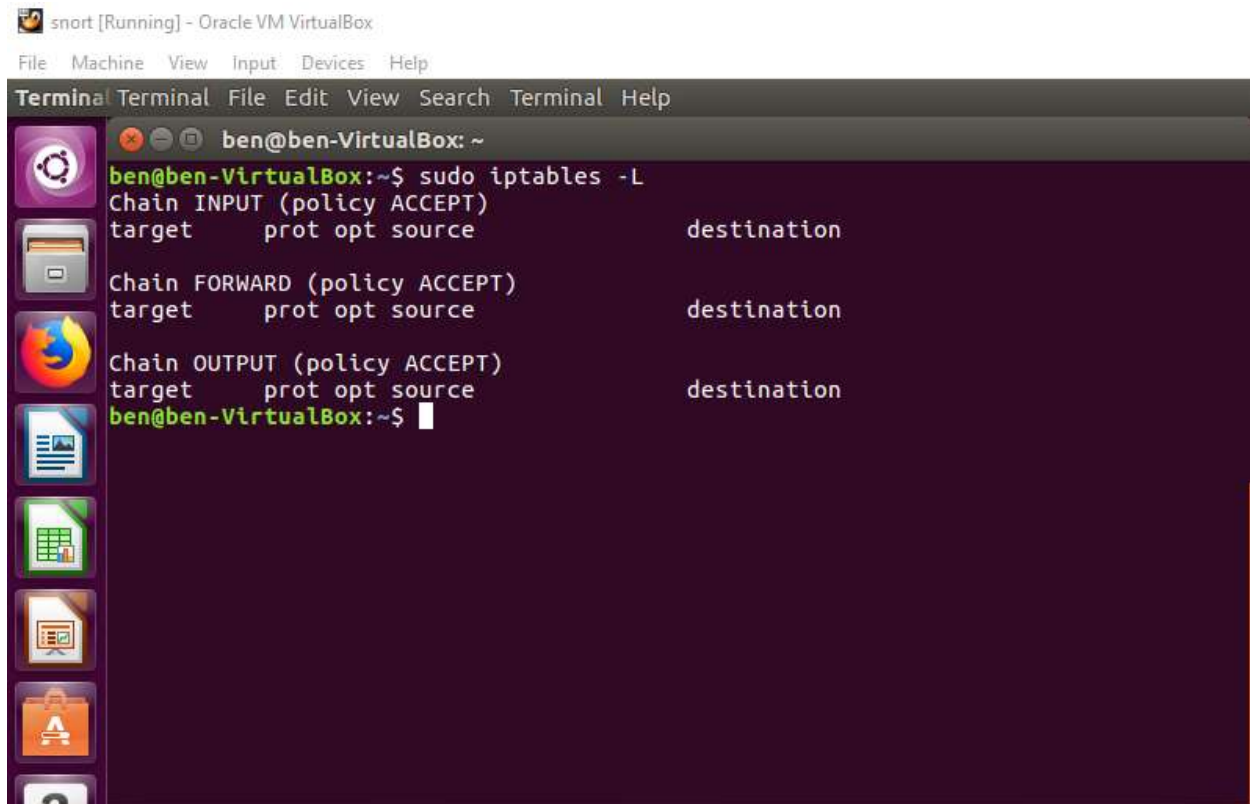


Figure 14: Regaining Capabilities By Using Set Capacity

Figure 14 shows an attempt was taken to regain capacity by using the command **set_cap_net_raw+ep /bin/ping** and **setcap cap_net_raw=ep /bin/ping** with no success.

4.0: Restricting Network Access

To complete this module iptables will be used which allows the system to block incoming, outgoing and internal packets.



```
ben@ben-VirtualBox: ~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ben@ben-VirtualBox:~$
```

Figure 15: IP Tables

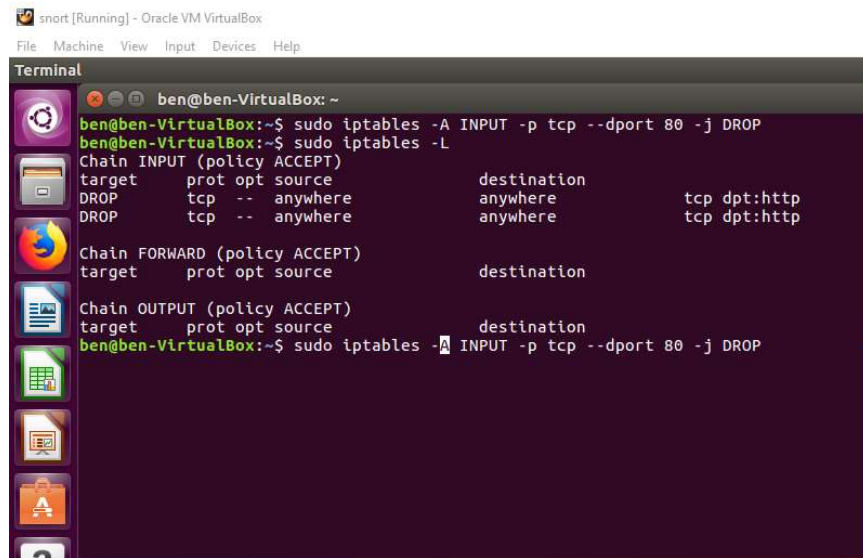
As we can see in Figure 15 iptables work by creating rules that govern Input chains, forward chains and output chains.

Input chains are rules that govern incoming packets to the system, forward chains are rules that dictate how packets are sent/received internally within the system and output chains are rules that govern how outgoing packets exit the system.

4.1: Blocking Network Traffic

4.1.1: Blocking Network Traffic -Protocols

By typing in the command `iptables -A INPUT -p tcp --dport 80 -j DROP` we have effectively dropped incoming TCP packets from entering the system on port 80.



```
snort [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
ben@ben-VirtualBox: ~
ben@ben-VirtualBox:~$ sudo iptables -A INPUT -p tcp --dport 80 -j DROP
ben@ben-VirtualBox:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target    prot opt source                destination
DROP      tcp  --  anywhere              tcp dpt:http
DROP      tcp  --  anywhere              tcp dpt:http

Chain FORWARD (policy ACCEPT)
target    prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target    prot opt source                destination
ben@ben-VirtualBox:~$ sudo iptables -A INPUT -p tcp --dport 80 -j DROP
```

Figure 16: Blocking Incoming TCP Packets on port 80 w

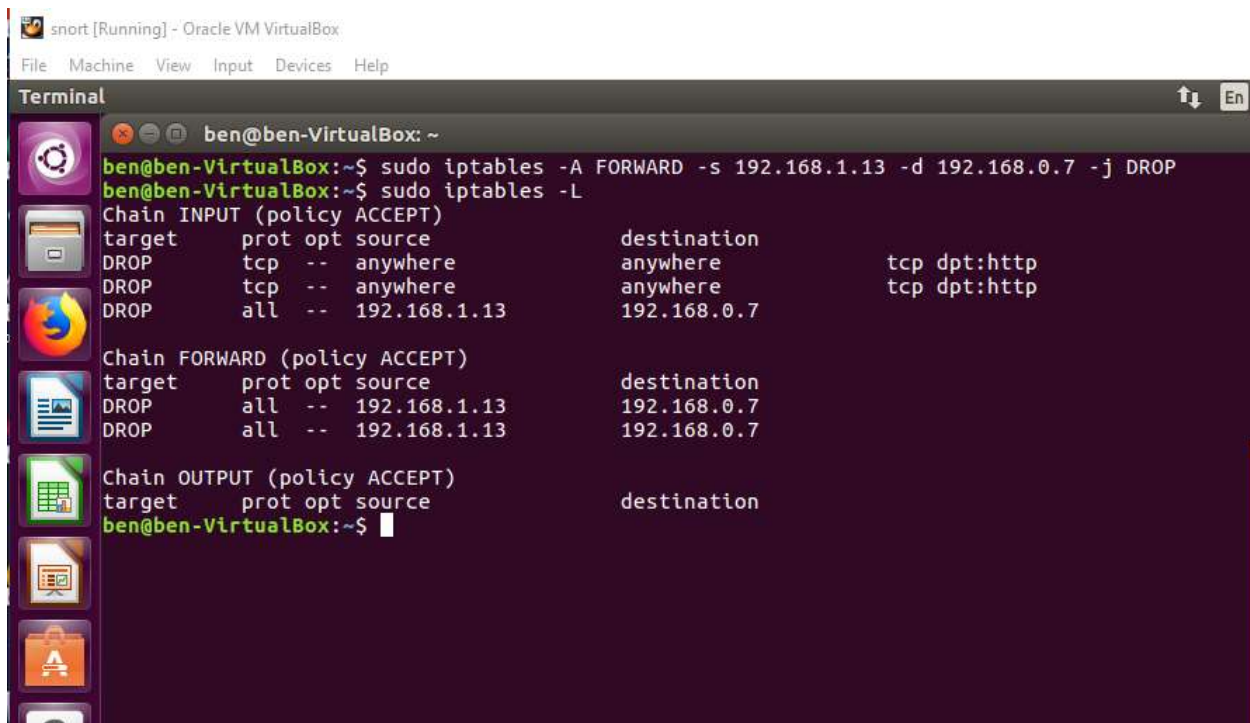
4.1.2: Blocking Network Traffic -IP

Similarly if the command `iptables -A INPUT -s 192.168.1.13 -d 192.168.0.7 -j DROP` is used we can drop incoming packets from entering the network at 192.168.1.13 and leaving the network at 192.168.0.7 (Figure 17).

4.1.3: Blocking Network Traffic -Application

Since forwarding chains are strictly meant to deal with internal communication it can be used to block internal application access.

To create a forward chain rule the command `iptables -A FORWARD -s 192.168.1.13 -d 192.168.0.7 -j DROP` is used (Figure 17).

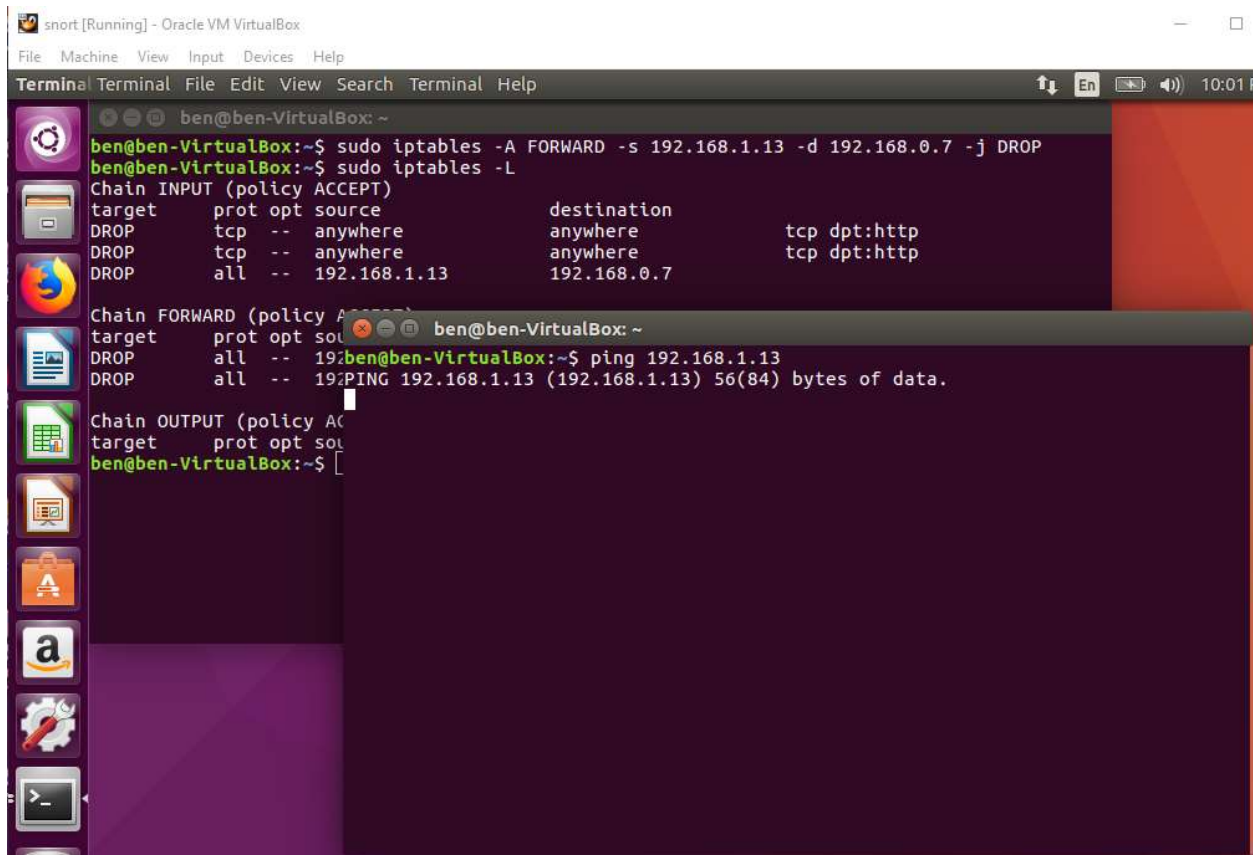


```
snort [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
ben@ben-VirtualBox: ~
ben@ben-VirtualBox:~$ sudo iptables -A FORWARD -s 192.168.1.13 -d 192.168.0.7 -j DROP
ben@ben-VirtualBox:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target    prot opt source                destination            tcp dpt:http
DROP      tcp  -- anywhere              anywhere               tcp dpt:http
DROP      tcp  -- anywhere              anywhere
DROP      all  -- 192.168.1.13          192.168.0.7
Chain FORWARD (policy ACCEPT)
target    prot opt source                destination
DROP      all  -- 192.168.1.13          192.168.0.7
DROP      all  -- 192.168.1.13          192.168.0.7
Chain OUTPUT (policy ACCEPT)
target    prot opt source                destination
ben@ben-VirtualBox:~$
```

Figure 17: Forwarding Rules

As can be seen in the Figure above we now have rules for the INPUT, FORWARD and OUTPUT chains.

4.2: Testing Network Traffic



The screenshot shows a terminal window titled "short [Running] - Oracle VM VirtualBox". The terminal output is as follows:

```
ben@ben-VirtualBox:~$ sudo iptables -A FORWARD -s 192.168.1.13 -d 192.168.0.7 -j DROP
ben@ben-VirtualBox:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination            tcp dpt:http
DROP      tcp  --  anywhere              anywhere              tcp dpt:http
DROP      tcp  --  anywhere              anywhere
DROP      all  --  192.168.1.13          192.168.0.7
Chain FORWARD (policy ACCEPT)
target     prot opt source                destination
DROP      all  --  192.168.1.13          192.168.0.7
Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
ben@ben-VirtualBox:~$ ping 192.168.1.13
PING 192.168.1.13 (192.168.1.13) 56(84) bytes of data.
```

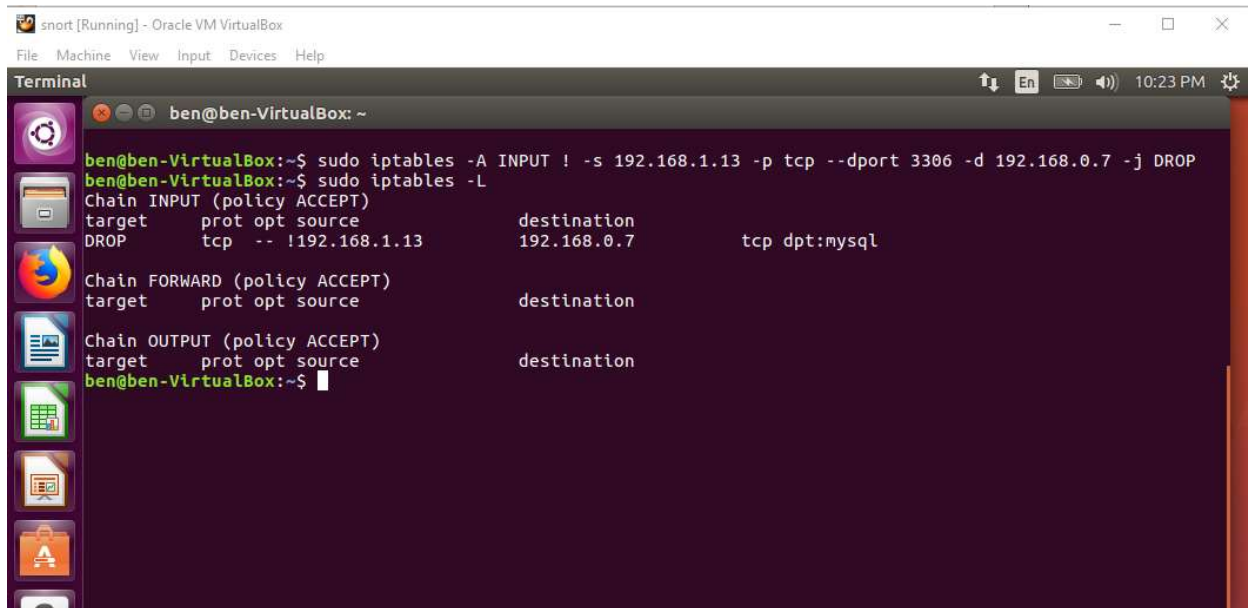
The terminal window also shows a sidebar with various application icons on the left and a status bar at the bottom.

Figure 18: Forwarding Rules

In the figure above we attempt to test the connection to **192.168.1.13**. As we can see the initially ping is successful since we did not block outgoing packets leaving the system. However incoming packets from **192.161.13** can not enter the system and after the initial ping no more packets can be received from the IP 192.168.1.13.

4.3: Implementing A Partial Block

A partial block is a bit tricky as it requires an understanding of what ports are being used by each application and what IP address is being used. In the example below only the SQL database from the IP 192.168.1.13 will be blocked, this is because only port 3306 on IP 192.168.1.13 was blocked from entering the system.



```
snort [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
ben@ben-VirtualBox: ~
ben@ben-VirtualBox:~$ sudo iptables -A INPUT ! -s 192.168.1.13 -p tcp --dport 3306 -d 192.168.0.7 -j DROP
ben@ben-VirtualBox:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target    prot opt source                destination
DROP      tcp  --  !192.168.1.13          192.168.0.7          tcp dpt:mysql

Chain FORWARD (policy ACCEPT)
target    prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target    prot opt source                destination
ben@ben-VirtualBox:~$
```

Figure 20: Partial Blocking

By typing in the command **iptables -A INPUT ! -s 192.168.1.13 -p tcp --dport 3306 -d 192.168.0.7 -j DROP** we effectively allow every process using the IP 192.168.1.13 to enter the system for the exception of MySQL database traffic.