COMP 4108 – COMPUTER SYSTEMS SECURITY

Assignment 1

Student Name: Ben Cendana
Student #: 100811212
Date: January 31 2018

Table Of Contents

Preface

References to each question are available in the Bibliography page.

1.0: Kernal Abstractions - Unix UID and GIDs

The label User ID and Group ID can be applied at the top of the architecture where the User Program/Applications are run. However the labels first get applied much deeper at the Kernel Level where the processes are run.

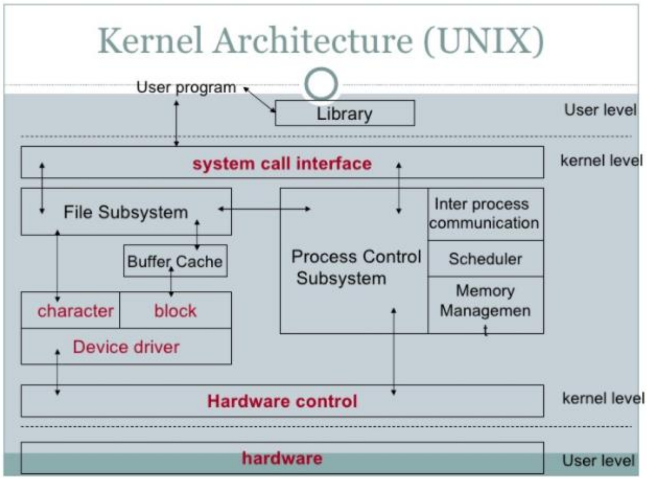


Figure 1: The UNIX Kernel Architecture.

As is demonstrated in Figure 1: The UNIX kernel Architecture we can see there exists several processes within the Kernel Level. Since we our interested in processes that deal with User ID's and Group IDs particular interest is the File Subsystem and the System Call Interface which are located within the kernel.

1.1: Kernel Abstractions – File Subsystem

The File Subsystem contains a system called inodes, the inodes keep a record of every file and the file ownership. Therefore if a file does not belong to a particular owner or group it will be the file subsystem that keeps a record of which files should be accessible to which users or groups.

1.2: Kernel Abstractions – System Call Interface

The System Call Interface is used to: 1) Interact with the user via a Unix Shell 2) Run System calls within the kernel. When the System Call Interface is used in conjunction with the file inodes it can prevent processes from executing in the Process Control Subsystem if the user attempts to run a process with out the proper valid UID or GID.

1.3: Kernel Abstractions – Conclusion

Thus it can be argued that since the File Subsystem keeps track of each file an there respective owners, the System Call Interface interacts with both the user and the Process Control Subsystem which is used to run the processes, its actually at the File Subsystem and System Call Interface are where the labels are actually first applied and then continues up to the top of the Unix Architecture.

2.0: Unix File Permissions

File permissions is Unix method to provide some security to critical files, these could include operating system files. However file permissions can be bypassed by taking advantage of the concept of ***other permissions*.**

The concept of other permissions stems from the idea that each file has set permissions for its owner (UID), the group (GID) and how other users can access the file or directory. In essence *other permissions* are the set permissions for users that are neither the owner nor a member of the user group.
Since outside users have there own set permissions for the file or directory they can change access by using the change mode or as its commonly referred to the **chmod** command**.**

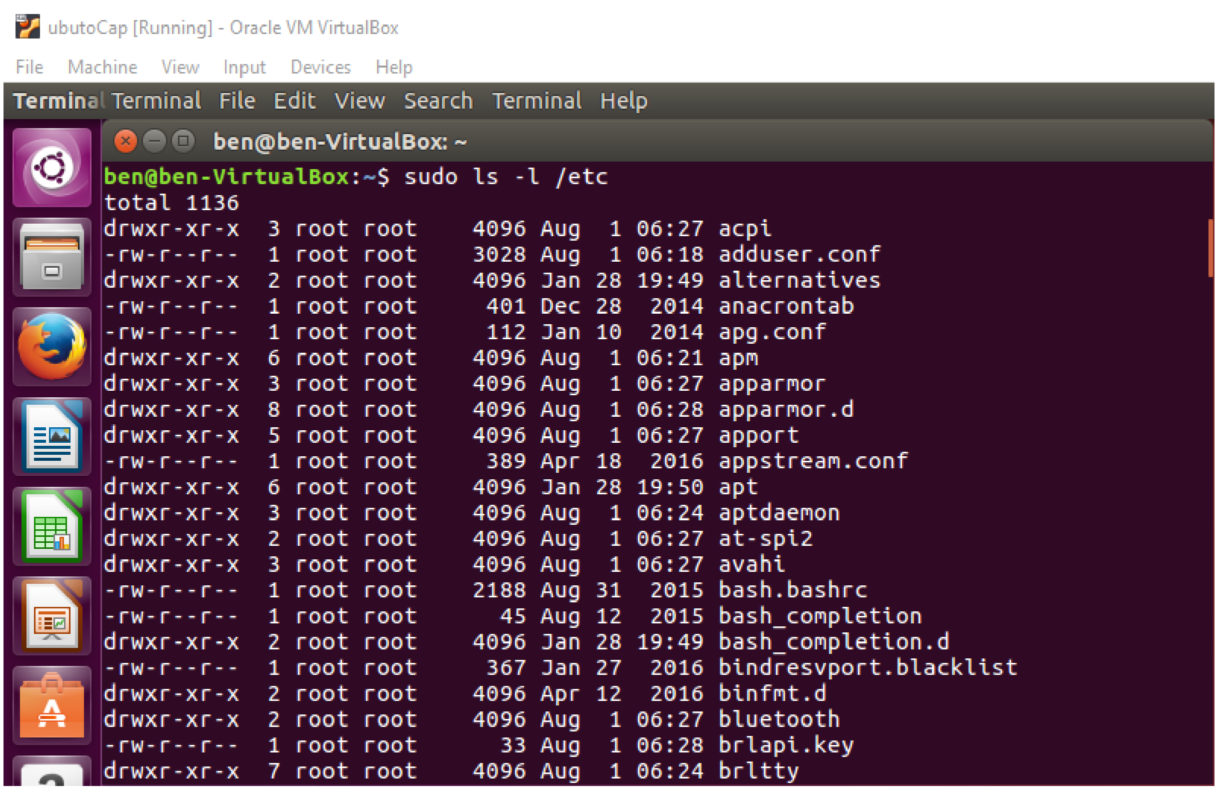


Figure 2: Processes With Only Root Permissions

As is demonstrated in Figure 2 the root /etc folder is only accessible if the user has the correct UID and GID access.

2.1: The Chmod Command

The change mode or **chmod** command provides the ability to change permissions in two ways Absolute and Symbolic.

2.1.1: The Chmod Command (Absolute Mode)

Absolute permission changes work by changing the assigned permission bit permission based on a number between 0 – 7.

Number	Description	Ref
0	No Permission	---
1	Execute Permission	--x
2	Write Permission	-w-
3	Execute and Write permission	-wx
4	Read Permission	r--
5	Read and Execute Permission	r-x
6	Read and Write Permission	rw-
7	All Permission	rwx

Table 1: Changing Permission Bits

Using the information in Table 1: Changing Permission Bits we can change permissions by typing in the command : **chmod ### filename or directory.**

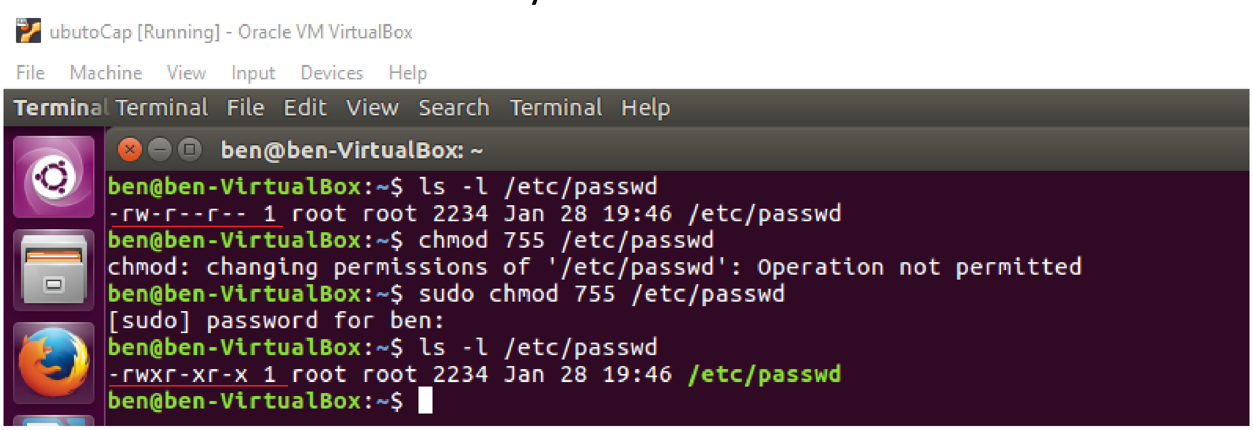


Figure 3: Using the Chmod 755 /file/passwd to add rwx and r-x permissions

As an example of how this works pay attention to the write permissions in Figure 3, notice how the /etc/passwd file initially only had -rw-r - -r permissions (read and write permissions). After inputting the command **Chmod 755 /file/passwd** we changed the permissions and were given permissions to execute the file.

2.1.2: The Chmod Command (Symbolic Mode)

The Symbolic Mode operates similar to absolute mode the only difference is instead of using numeric numbers symbolic mode uses arithmetic operators.

Operator	Description
+	Adds The Designated Permission
-	Removes The Designated Permission
=	Sets The Designated Permission

Table 2: The Symbolic Chmod Commands

Using table 2 as a guide we can change the permissions by using the command:

chmod o#wg file
chmod o#x file
chmod g#rx file

Figure 4 below demonstrates how typing in the command **chmod o+x /etc/shadow** added the execute permission to the file.

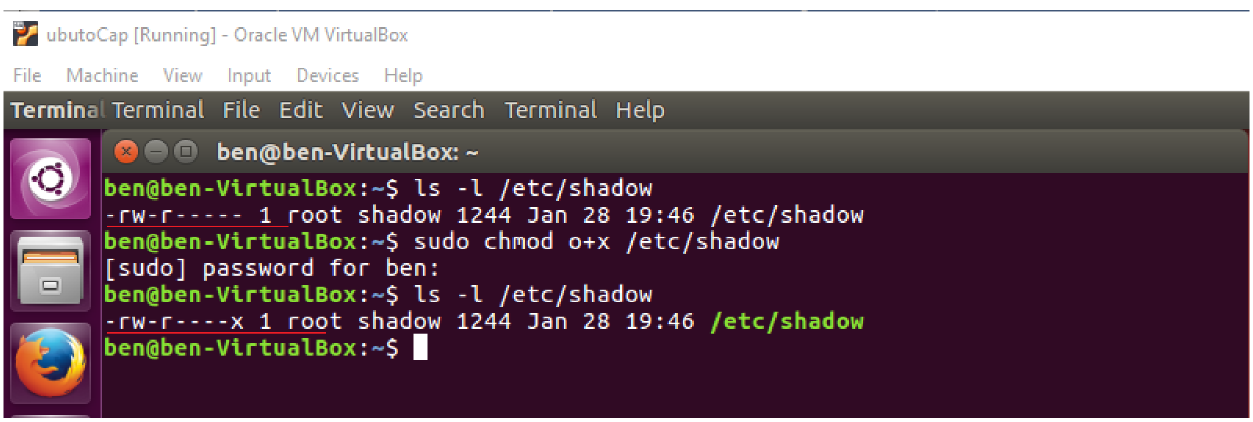


Figure 4: Using the command chmod o+x /ect/shadow to add execute permissions

3.0: Read, Write and Execute Permissions

In the preceding section we saw how file permissions can be changed to read, write and or execute, in this section we will describe each and how they perform.

We can determine what the file/directories permissions are by typing in the command **ls -l** (Figure 5).

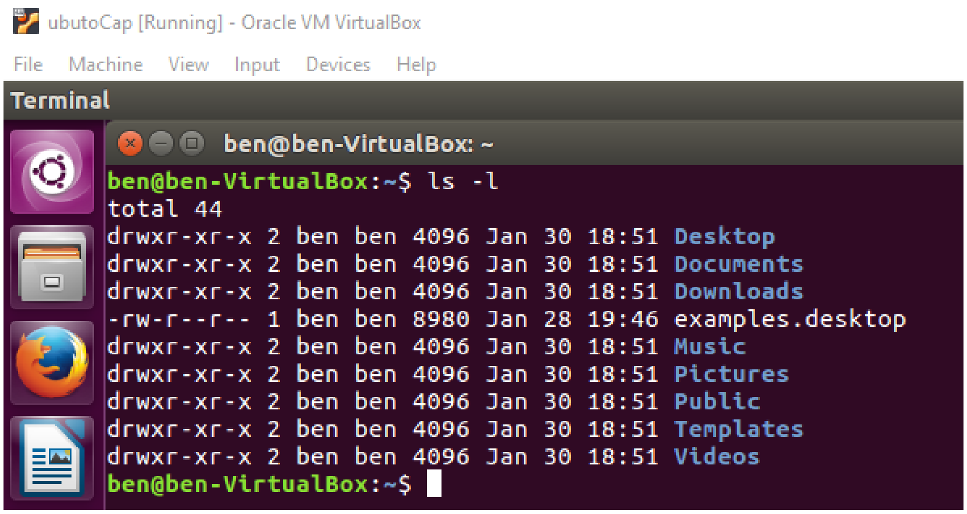


Figure 5: Displaying The File Permissions

3.1: Read Permissions

As the name suggests read permissions indicate that file/directory can be opened and viewed this is indicated by the letter ***r.***

3.2: Write Permissions

Write permissions allow the user and root to modify the file/directory this is indicated by the letter **w.**

3.3: Execute Permissions

Execute permissions allow the user and root to run the file/directory this is indicated by the letter **x.**

3.4: Verification

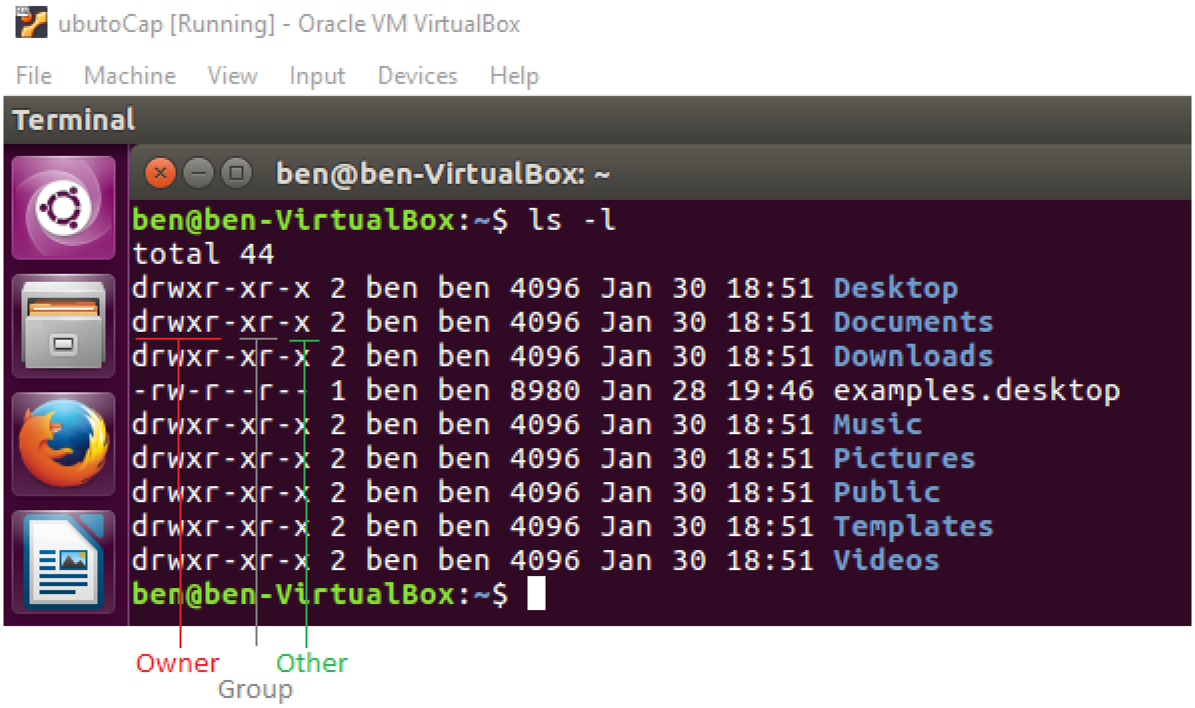


Figure 6: Displaying The Read, Write and Execute Permissions By User

Verification can be done by typing in the command **ls -l**, as we can see in Figure 6 each user has a different set of permissions. In the example above *other* users only have execute permissions while users that are member of that *group (GID)* have read and execute permissions.

4.0: Sticky Bit

4.1: Sticky Bit: Prevent Deletion

The phrase stick bit refers to a method of preventing a critical file or directory from being deleted or used. In most cases this is done to protect system files were only users that have root privileges, or the user is owner of the file/directory or the user has privileges to delete the file. To declare a file or directory as a sticky bit the command **chmod 1753 file** or **chmod +t file** is used (Figure 7).

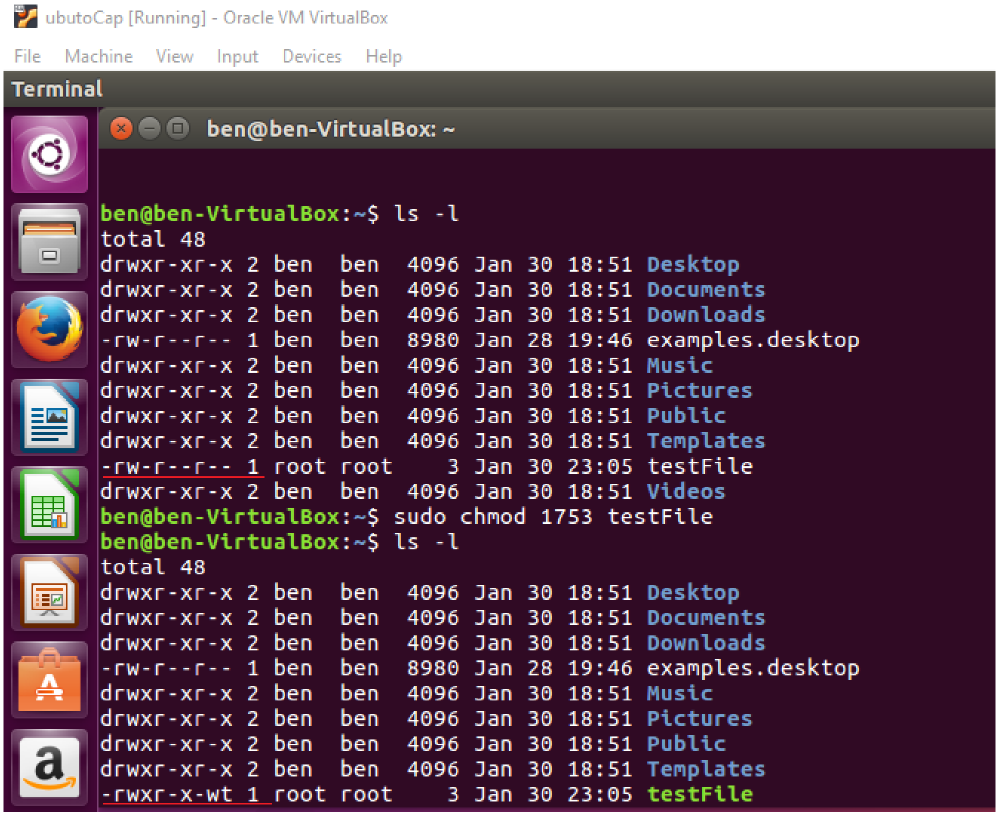


Figure 7: Declaring A StickyBit File

As we can see in Figure 7 after applying the command **chmod 1753 testFile,** the file testFile now has a **t** added indicating it's a sticky file. **Note**: a sticky file or directory can be indicated by either a letter **t** and **s.**

4.2: Sticky Bit: Temporary Root Access

Another reason for using sticky bits is to allow users temporary root privileges to execute a particular file/directory. The user may gain root access, but only for that file/directory.

4.3: Sticky Bit Attacks

As previously mentioned the primary reasons for using a sticky bit is to prevent unauthorized users from accessing critical files and directories. By doing so we prevent critical files such as system files from being modified, replaced or overwritten.

5.0: Setuid Binaries

5.1: Setuid Binaries - What Are Setuid Binaries

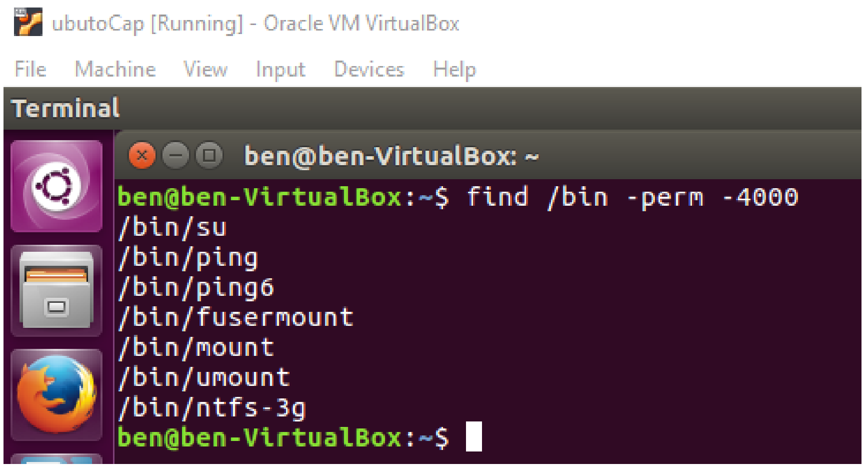


Figure 8: Displaying Setuid Binaries

The setuid binaries are essentially sticky bit directory files that contain critical system files that can be accessed only by users who have the read, write and execution rights. A user who has read, write and execution rights to setuid directories will inherit root privileges, essentially changing the users capabilities to root or super root status.

5.2: Setuid Binaries – Why Are They Important

If a user gains or has access to a setuid binary he/she gains access to all the critical system files as they now become root or super root. Similarly they can also change the permissions of a set group id (SGID) and give members of the group id access as well. From a security perspective this can increase the risk of users accessing unauthorized system files.

5.3: Setuid Binaries – Security Risks

As we can see in Figure 8 the directory /bin/ping permissions were changed to read, write and execute for both the SUID and GUID. This means the file directory can now be tampered with. All the other setuid binaries share this risk.

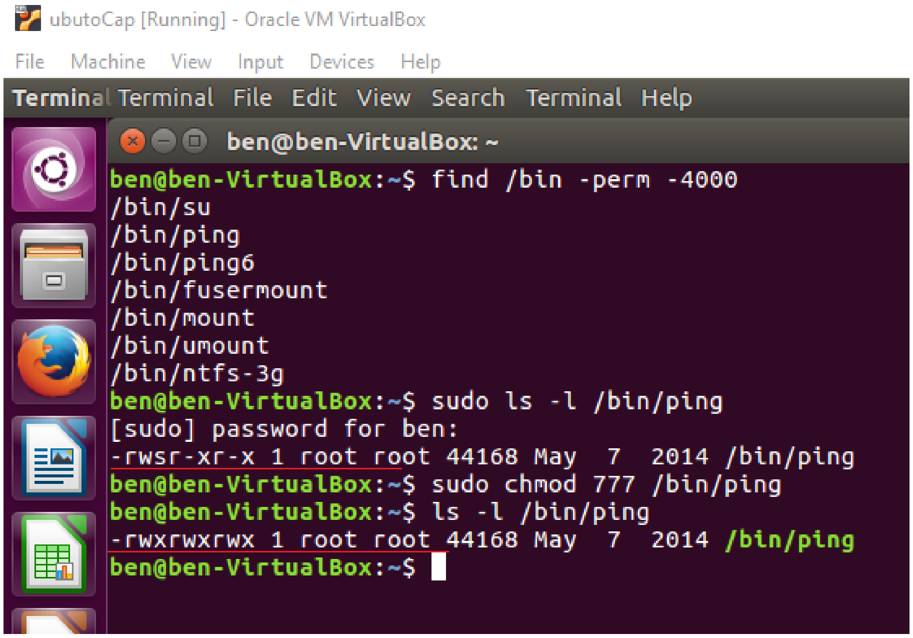


Figure 8: Changing The Permissions Of the /bin/ping Binary.

Once the permissions have been changed any user can now access the setuid binary, then become root and tamper with the system files.

6.0: Potential Vulnerability and Attack Of A Set UID

As an example of a vulnerability and attack we will change a users password by accessing the /etc/passwd and /etc/shadow setuid binaries.

The first step is to change the set permissions with the chmod command to allow any user that becomes root when they access the setuid binaries full read, write and execute capabilities.

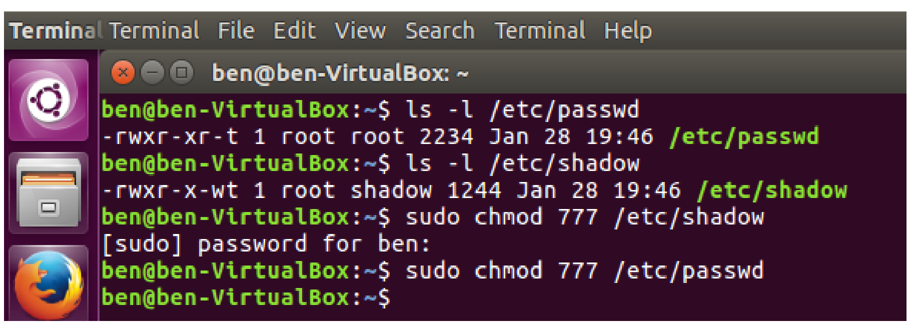


Figure 9: Changing The Permissions Of the /bin/ping Binary.

As we can see in Figure 10 below the read, write and execute permissions have been changed for the /etc/passwd and /etc/shadow binaries.

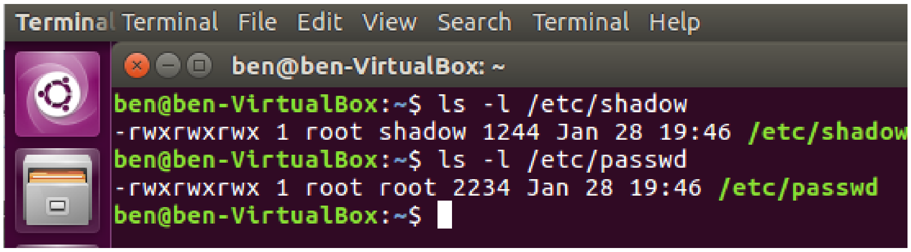


Figure 10: Displaying The Change In Permissions

Since the binary files are now accessible a simple script called rootScript.c can access the binaries and become root simply by making a system call to the corresponding files.

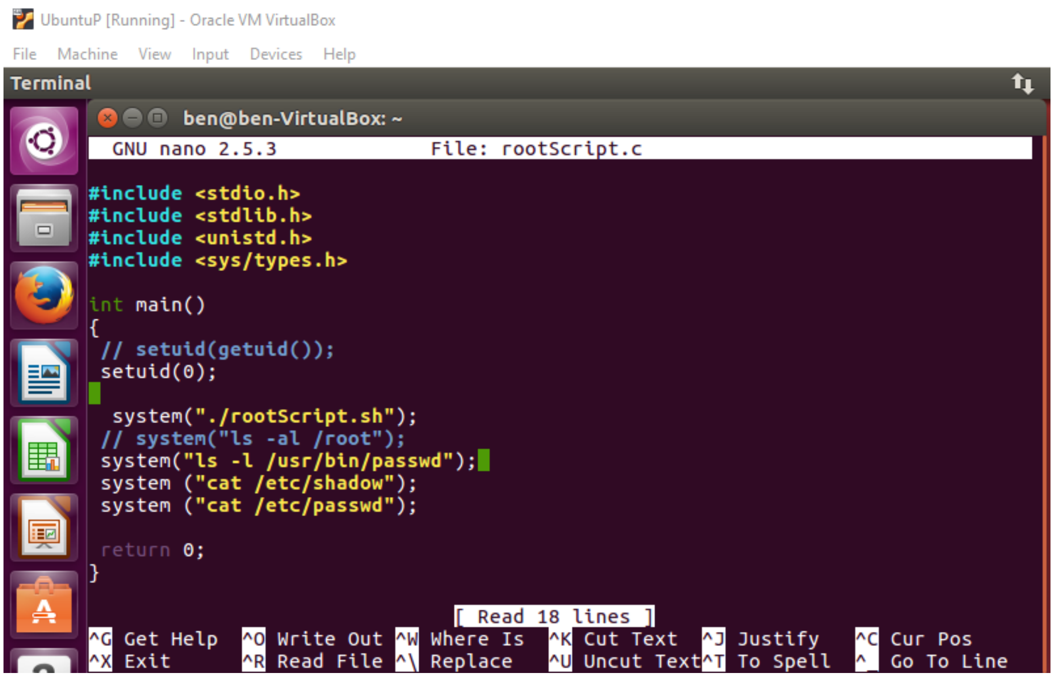


Figure 11: The rootScript.c

As we can see in Figure 12 and Figure 13 shown below, we were able to view all the user passwords and change them by using rootScript.c.

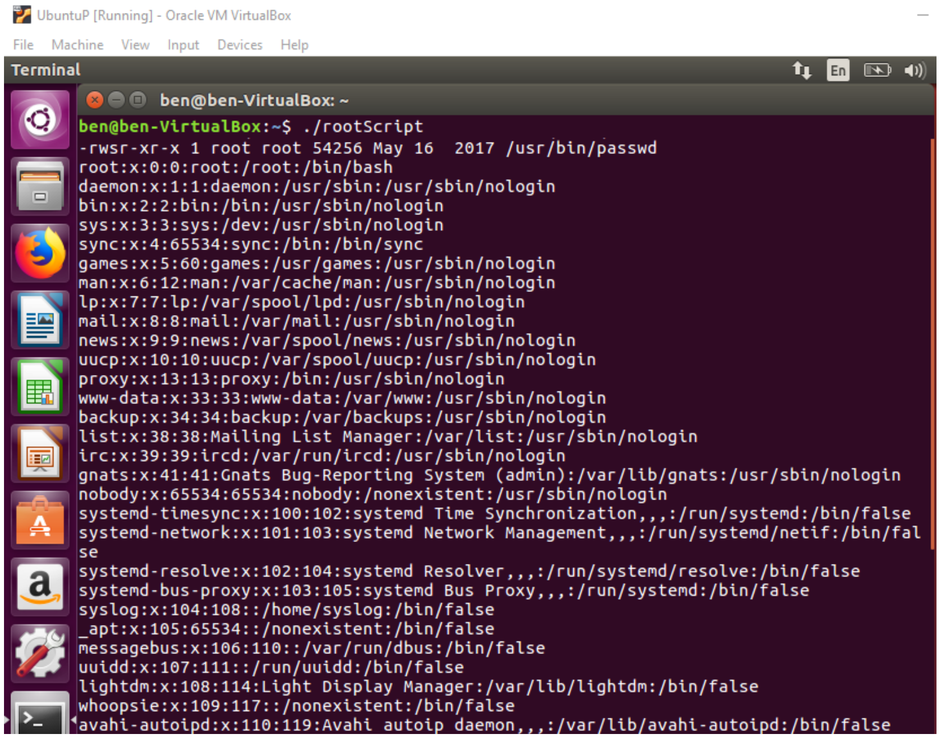


Figure 12: Displaying The User Passwords

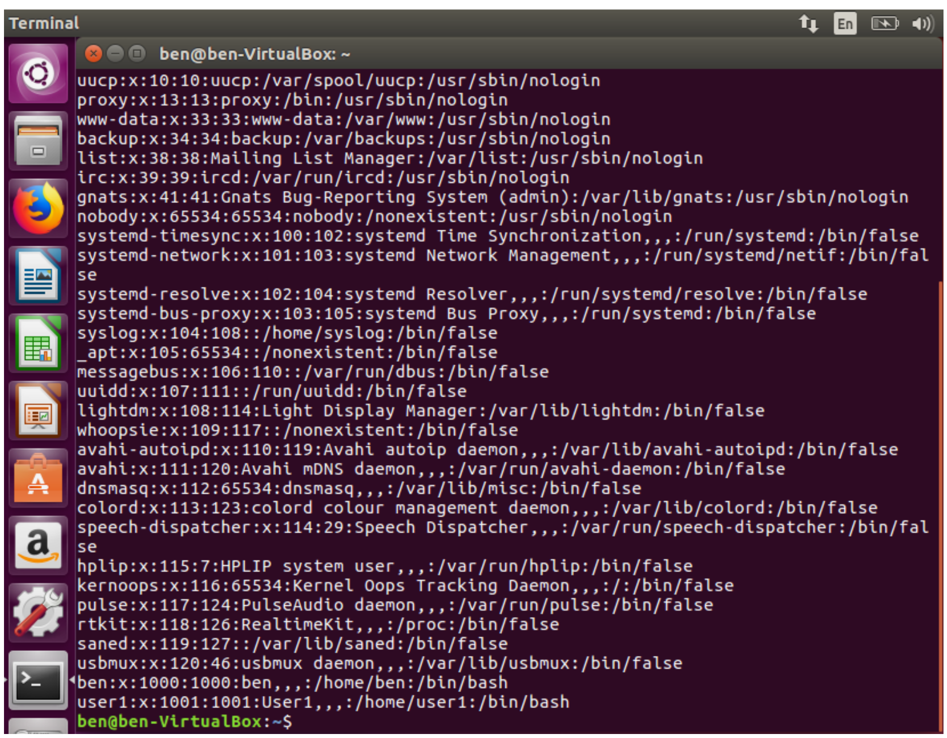


Figure 13: Changing The User Passwords

Again, this is just one example of how an attack could be carried out there exists many other ways of exploiting setuid. But it demonstrates the risks setuid binaries have if they are tampered with and the wrong users are given higher privileges.

7.0: TCP Wrappers and Firewalls Protection

Figure 14: An Artist Depiction Of A Firewall

As the name suggests a firewall or TCP wrapper acts as a barrier between the network and other networks the user(s) may connect to. TCP wrappers and firewalls protect the network by creating rules that govern the network. These rules include blocking incoming packets from specific IPs to filtering protocol access of incoming/outgoing traffic from specific ports. The primary reason why these rules are in place is to prevent threats from malicious code, these include viruses and malware that may enter the network or machine.

8.0: TCP Wrappers vs Firewalls – Technical Differences

8.1: Firewalls

A host based firewall is a firewall that is installed on every machine and monitors incoming packet traffic on the individual machine.

8.2: TCP Wrapper

Unlike host based firewalls a TCP wrapper is essentially an access control link (ACL) library that controls UNIX processes and there user privileges. As an example suppose that root only access is maintained throughout the system. As such the ACL will prevent outside network traffic from accessing files/directories and even processes from being executed because it does not have root privileges.

9.0: VPN System Changes and Behaviour

When a user connects to a virtual private network the user may experience changes in the delivery of content he/she may have access to. For example in countries such as China where certain websites are censored or blocked the user may now be able to access full content. Moreover the VPN will also assign the user a new IP which will replace there previous IP.

The user can verify that there connected to the VPN once the *handshake* takes place, this is when both the VPN and user exchange there public keys. Once the public keys have been authenticated the *handshake* confirms that the VPN is indeed the VPN and the user is indeed the user.

10.0: VPN Organization Security

One of the primary reasons why an organization may setup a VPN is to allow its employees to securely connect to the organizations network while being connected to an insecure local network. The local network maybe insecure but the point to point connection between the employees machine and the network is secure because both the network and client are authenticating each other via there public keys.

The disadvantage to a VPN is once the network becomes infected it will spread to every user that connects to the VPN. An excellent example how this may take place is lets assume a client starts to open there outlook and view all there incoming emails. The client may not be aware or forgets to connect to the VPN, as the user opens there emails they receive a malware which infects there machine.
The client then realizes there not connected to the VPN and decides to connects but while doing so they infect the VPN network.

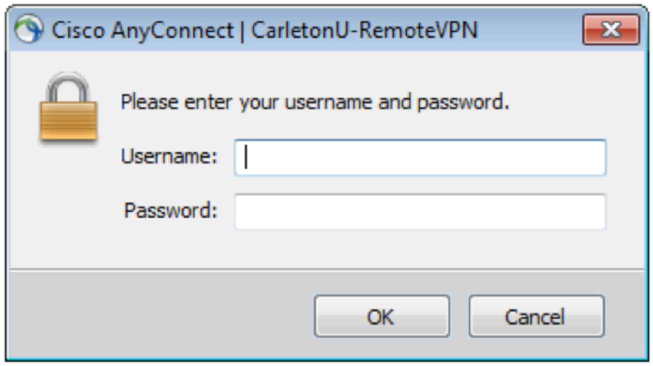
Moreover the VPN does not completely solve the problem of public key authentication, the connection between the user and the VPN maybe secure but once traffic leaves the VPN on the behalf of the user the connection again becomes insecure. In essence a VPN only serves as a intermediate between the user and accessing the world wide web.

11.0: Carleton VPN

The primary VPN used by the Carleton University network is AnyConnect.

After initial installation the AnyConnect VPN works by sending a digital certificate which is then installed onto the users browser.

Some of the advantages to using the VPN are accessing the OpenStack Dashboard off campus, accessing your personal network drives and the department network drives.



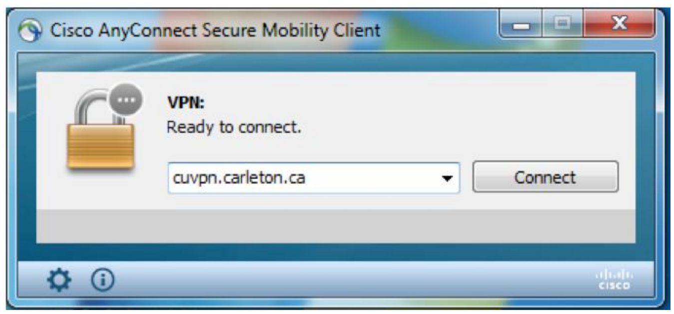


Figure 15: Connecting With The AnyConnect VPN

12.0: Bibliography

Section 1.0

Reys, Gled. "Hard Link." Unix Tutorial, 4 July 2014, www.unixtutorial.org/glossary/hard-link/

Shah, Chandni. "Ch2." LinkedIn SlideShare, 3 May 2012, www.slideshare.net/chshah2486/ch2-12781554

Arnold, Ken. "UNIX System Calls." UNIX System Calls, www.di.uevora.pt/~lmr/syscalls.html
Ramamurthy, B. "UNIX Process Control ", 3 Sep 2003,
https://www.cse.buffalo.edu/~bina/cse421/spring2003/UnixProcessControlFeb5.pdf

Section 2.0

tutorialspoint.com. "Unix / Linux File Permission / Access Modes." www.tutorialspoint.com, www.tutorialspoint.com/unix/unix-file-permission.htm

Section 3.0

UNIX: The Complete Reference ; Second Edition,
Kenneth H. Rosen. Douglas A. Host. Rachel Klee. Richard R. Rosinski - McGraw-Hill - 2007

Section 4.0

UNIX: The Complete Reference ; Second Edition,
Kenneth H. Rosen. Douglas A. Host. Rachel Klee. Richard R. Rosinski - McGraw-Hill - 2007

Section 5.0

The Linux Juggernaut "What Is Suid And How To Set Suid In Linux/Unix." www.tutorialspoint.com, www.tutorialspoint.com/unix/unix-file-permission.htm

Occupytheweb, and WhiteRabbit. "Hack Like a Pro: Finding Potential SUID/SGID Vulnerabilities on Linux & Unix Systems." WonderHowTo, WonderHowTo, 7 Nov. 2014, https://null-byte.wonderhowto.com/how-to/hack-like-pro-finding-potential-suid-sgid-vulnerabilities-linux-unix-systems-0158373/

Linux Capabilities and Ping. 30 May 2016, www.nixetc.co.uk/2016/05/30/linux-capabilities-and-ping/

Kalafut, Andrew. CIS 458 Lab 4: Capabilities in Linux. 2013, www.cis.gvsu.edu/~kalafuta/cis458/f16/labs/lab4.html

Section 6.0

Based on Experience 2

Section 7.0

Based on In class lecture

Section 8.0

"Host-Based vs Network-Based Firewalls." Wideband, 10 Mar. 2017, www.wideband.net.au/blog/host-based-vs-network-based-firewalls/.

nixCraft , "Explain: Linux and UNIX TCP Wrappers – Find Out If a Program Is Compiled With TCP Wrappers" February 24, 2014, https://www.cyberciti.biz/faq/tcp-wrappers-hosts-allow-deny-tutorial/

Rouse , Margaret. "What Is Access Control List (ACL)? - Definition from WhatIs.com." SearchSoftwareQuality, www.searchsoftwarequality.techtarget.com/definition/access-control-list.

Section 9.0 - 10

Based on In class lecture

Section 11.0

"Information Technology Services." Help Centre, 27 Jan. 2017, https://carleton.ca/its/help-centre/remote-access/

"OpenStack." How to Get an Account on SCS OpenStack, School of Computer Science, https://carleton.ca/scs/technical-support/scs-open-stack/

"Information Technology Services." VPN for Windows 10, 27 Jan. 2017, https://carleton.ca/its/help-centre/remote-access/vpn-for-windows-10/