Access Control Project

Charan Balavenkatigari

Creating Users

User Creation Method/Evidence

Before creating a user, the `sudo su` command was entered to log in as the root user, providing administrative privileges necessary for managing the file directory (Purdue University. 2016). To create users, the command `adduser [username]` was used for every user as shown in Figure 2. There are 2 methods to add users to the file, but utilizing the `adduser` command is preferred as it is more simple, and follows the standard username conventions. For instance, it ensures the username firs the system requirements which enhances consistency and security. While, the other method, `as--force-badname`, can cause confusion and security problems in the future as it doesn't follow the same criteria (Wu, 2019). During this process, each user was assigned a common password. This process was repeated 9 times for each employee.

Figure 1: Evidence of Switching to Root User

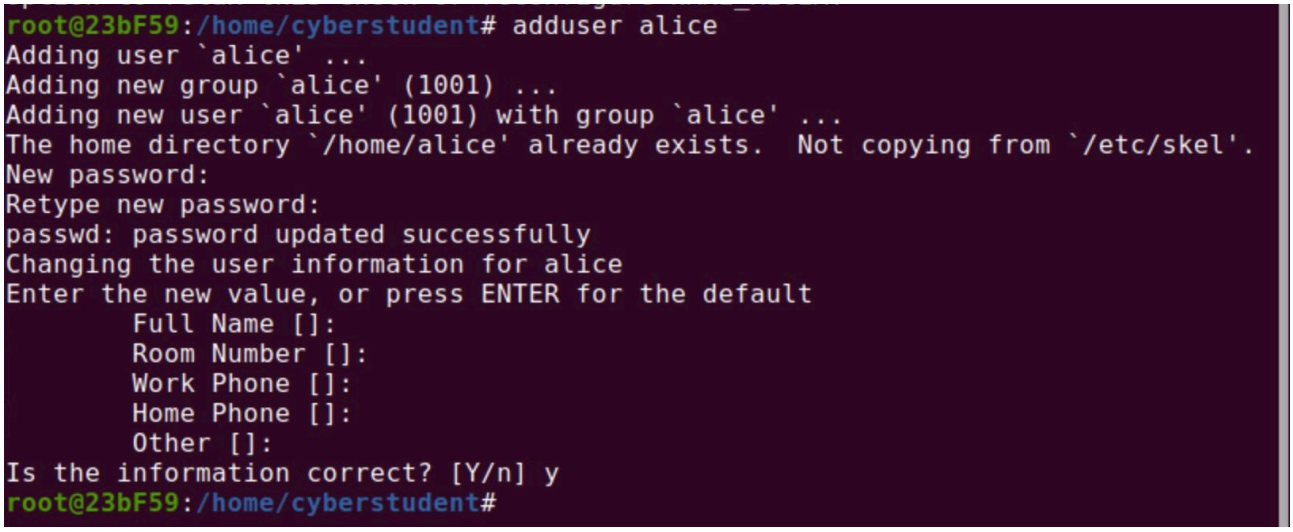

Figure 2: Evidence of Creating a User

User Creation Audit/Evidence

After adding users to the system, it's essential to verify that each user has been created with the correct settings. This includes checking usernames, home directories, and groups. Figure 3 displays a list of the 10 employees from Hands Hardy Hardware who are now users proving they are in the correct group. This user audit evidence was found by utilizing the `tail -10` command. The purpose of the tail command is to output the last 10 lines of a file (Purdue University. 2016).

Figure 3: Evidence of User Creation Audit

Grant Admin Method

To grant the created users admin access, the first thing to do was use the `sudo usermod -aG sudo [username]` command to grant administrative privileges to the selected users. The -a flag stands for append and is used with the -G flag to add a user to a group without removing them from other groups (Manage User and Groups in Linux - TecAdmin, 2018). Then, -G is utilized to specify the groups to which a user belongs (Manage User and Groups in Linux - TecAdmin, 2018). This method added users to the sudo group, allowing them to execute the same privileges as a superuser (How to Grant Admin Privileges to a User in Linux, 2023). Figure 4 is an example of how Alice acquired admin privileges. This same process applied to Alice, Bob, and Hands. Since Alice and Bob are a part of the IT and Security team, they require access to manage security and networks and are required to manage other users. Hands, the CEO, was granted administrative privileges, enabling them to oversee the company effectively (Hands, N. 2024). This access allowed them to view, manage, or approve system changes and sensitive information when necessary. After successfully creating our designated users and giving certain ones access control, the `grep sudo /etc/group` command, as shown in Figure 5, listed all of the users with admin privileges. The purpose of the grep command is to search for patterns in

files (Purdue University. 2016). Then, the `/etc/group` was the file that contained the users and was being searched for the pattern (sbiradar, n.d.).

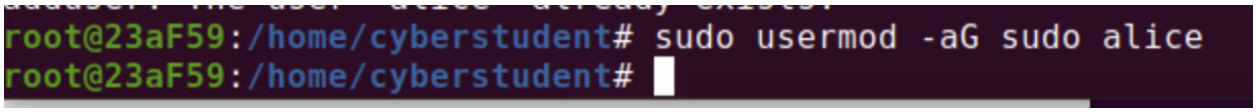

Figure 4: Evidence of Granting Alice with Admin Privileges

Figure 5: Evidence of Granting all Users with Admin Privileges

Password Force Method/Evidence

After the users were created, temporary passwords were assigned as a requirement during the creation process. The command `sudo tail -9 /etc/shadow` was used to see the password properties for all users in Linux (Linux Commands - GeeksforGeeks, 2019). As mentioned earlier, `sudo,` allows the command to function as a superuser. The `tail` command outputs the last part of a file, and in this case, `-9` specifies the last 9 lines of the file. Then, `/etc/shadow` file stores hashed password information. Figure 6, reveals the password can be changed basically anytime, specifically between 0 and 99999 days. This is shown when it displays `0:99999`. The `:7` means there will be a warning given 7 days before the password is about to expire.

Figure 6: Evidence of Prompting Users with Password Change at Next Log-In

Trent and Malory's Justification:

Role-based access control is when companies give access to employees based on their role, however, not every employee is given the same access (Zhang, 2023). Trent's role in this company was Research & Development (R&D), and Malory had an accounting role. The principle of least privilege means users who need to access certain data can access it while those who don't need to cannot (Rosencrance, 2023). This principle was the reason both Trent and Malory were not granted admin privileges as their roles don't need admin privileges to perform their tasks. If Trent or Malory were granted unnecessary privileges, it could lead to potential risks, such as the ability to add or remove users and assign permissions to others, thereby increasing the likelihood of data breaches. This VM had access to configuration files and could have important data/authorizations which in the wrong hands can cause financial and asset loss in the company if breached. Due to the principle of least privilege Malory and Trent should not have access and it should only be in the hands of those who directly manage this VM which are the IT department and the CEO.

Permission Bits Review

Permission Bits Permutations

In Linux, each object has permission bits attached that determine the permission of the file (Cipriani, 2020). Figure 7 summarizes every combination of permission bits and explains below:

Figure 7: Permutations of Special Permission Bits

Binary Bit Permutations	**Octal Bits**	**Qualitative Description**
000	0	No read, write, or execute permission is given to the user. No users, except root, would be able to read the file.
001	1	Only execute permission is given to the user. The user could run the file.
010	2	Only written permission is given to the user. The user could make edits to the file.
011	3	Write and execute permission are given to the user. The user could only run and write to the file.
100	4	Only read permission is given to the user. The user could only read the file.
101	5	Read and execute permissions are given to the user. The user could only read and execute the file.
110	6	Read and write permission is given to the user. The user could only read and write to the file.
111	7	All read, write, and execute permissions are given to the user. The user could perform all actions on the file, including reading, writing, and

		running the file.

Special Permission Bits

In Linux, special permissions extend the read, write, and execute permissions to a fourth degree of access control. The three types include Set User ID (SUID), Setgid (SGID), and the sticky bit. SUID allows users to execute a file with the permission of the file's owner. SGID enables users to run a file with the permissions of the group that owns the file. Then, Sticky Bit Restricts file deletion within a directory so that only the file's owner can remove it. These special permissions enhance security by providing additional control over how files and directories are accessed and modified (Cipriani, 2020). Figure 8 below provides each special permission bit with more detail below:

Figure 8: Special Permission Bits

Special Permission Bits	**Binary Permutations Bit**	**Octal Bits**	**Qualitative Description**
Setuid (SUID)	4000	4	Allows the file to be executed with the permissions of the file owner, rather than the user running the file.
Setgid (SGID)	2000	2	Makes it so that the user executes the file as the owner group. Once configured, all of the files inside the directory will be owned by the group.
Sticky Bits	1000	1	Ensures that only the file's owner, the directory's owner, or the root user can delete or rename files within that directory.

Creating Groups

Group Creation Method

Figure 9 demonstrates how the `sudo groupadd Marketing` command was used to create a group named Marketing (sbiradar, n.d.). A group can be created and given a distinct group ID by using the command `groupadd [groupname]`. This was helpful since it enabled a group that can assign selected users and identify them with an ID that will make it simple to locate and edit in the future.

```
root@23aF59:/home/cyberstudent# sudo groupadd Marketing
groupadd: group 'Marketing' already exists
root@23aF59:/home/cyberstudent#
```

Figure 9: Evidence of Creating a Group

Group Membership Evidence

To add users to the Marketing group, the `usermod -aG Marketing[employee name]` command was used (Avi, 2024). The `-aG` tail is useful to ensure an efficient and secure management of user groups on the Linux system. The chosen employees were Carol, Victor, and Hands. The reason Carol and Victor were perfect for this position was that Carol works in Sales and Victor was in customer service. These were two job roles that were relevant in marketing because they were able to understand customer experience and revenue. Additionally, Hands was added as he was the CEO and lead in marketing, so he deserved access to all groups. Figure 10 confirms Carol, Victor, and Hands were in the group by utilizing the `tail -1 /etc/group` command (Avi, 2024). According to the table in the lab manual, the rest of the departments were organized correspondingly as shown in Figure 10.

```
root@24aF59:/home/cyberstudent# tail -9 /etc/group
Marketing:x:1014:carol,victor,hands
remoteusers:x:1015:alice,carol
IT:x:1016:bob,alice
Sales:x:1017:carol
Accounting:x:1018:malory
ResearchAndDevelopment:x:1020:trent,walter,hands
customerService:x:1021:victor
receptionist:x:1022:eve
CEO:x:1023:hands
root@24aF59:/home/cyberstudent#
```

Figure 10: Evidence of Finding Departmental Groups

Directory Structure Reasoning

Creating a group share file allowed easier access to those with the authorization to be a part of that file. This enabled the group file to be a part of a directory, resulting in better organization of files and folders. This was located in `/srv` in the desktop of the filesystem (Linux Commands - GeeksforGeeks, 2019). Additionally, allows better security as it can be set to those who get added to the directory. The number of folders added to the directory that was created was only one which had no child folders and was assigned to the Marketing group. A directory was created for each department group. The filenames had no spaces and the directory name itself was all lowercase.

Directory Creation Methods

Before creating the new directories, the command `cd /srv` was used to navigate to the /srv directory. The `cd` command, which stands for "change directory," enables switching between directories within the filesystem (Purdue University. 2016). Then, Figure 12 illustrates how the `ls -l` command listed the items in the current directories and showed the permissions in the /srv directory (Purdue University. 2016).

Figure 11: Evidence of Changing Directory

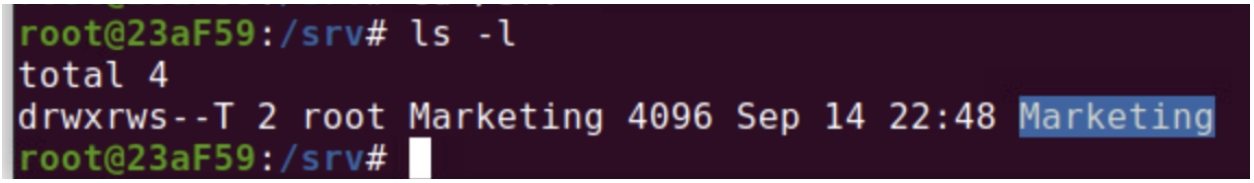

Figure 12: Evidence of Items and Permissions in the Directory

To create the Marketing directory, the `mkdir` command was utilized with `sudo mkdir /Marketing`, as shown in Figure 13. The `mkdir` command, short for "make directory," allows for the creation of directories in specific locations within the filesystem. Additionally, using the `-p` option with `mkdir` ensures that parent directories are created if they do not already exist (Linux Commands - GeeksforGeeks, 2019).

Next, Alice and Bob needed to make sure that their respective groups could access and use the shared resources by implementing group ownership (Hands, N. 2024). This means a shared folder needed to be created, so Figure 14 displays the `sudo mkdir -p /Marketing/sharedfolder` command. This command specifies the path and structure of

the new Marketing directory that was created. Figure 15 verifies the shared folder was created (Linux Commands - GeeksforGeeks, 2019). Then, the command `sudo chown :Marketing /Marketing` was used to change the ownership of the Marketing directory to the Marketing group, as shown in Figure 16 (Purdue University. 2016). The purpose of the `Chown` command was to change the ownership of a file to the user and group listed. This allowed the users in the Marketing group to access the directory based on the group permissions. Additionally, this enabled users to collaboratively share and manage files.

```
root@23aF59:/srv# sudo mkdir /Marketing
mkdir: cannot create directory '/Marketing': File exists
root@23aF59:/srv#
```

Figure 13: Evidence of Checking Directory Creation

```
root@23aF59:/srv# sudo mkdir -p /Marketing/sharedfolder
root@23aF59:/srv#
```

Figure 14: Evidence of Marketing Directory Creation

```
root@23aF59:/home/cyberstudent# ls -l /Marketing
total 8
drwxr-xr-x 2 root root      4096 Sep 18 20:54 sharedfolder
drwxrws--- 2 root Marketing 4096 Sep 13 14:06 sharedFolder
root@23aF59:/home/cyberstudent#
```

Figure 15: Verification the Shared Folder Exists

```
root@23aF59:/srv# sudo chown :Marketing /Marketing
root@23aF59:/srv#
```

Figure 16: Evidence of Changing Ownership of File to Admin

Evidence of File Structure

The tree command prints a tree diagram of the file structure from the directory (Purdue University. 2016). Figure 17 illustrates the tree diagram of the /srv directory in the Marketing group.

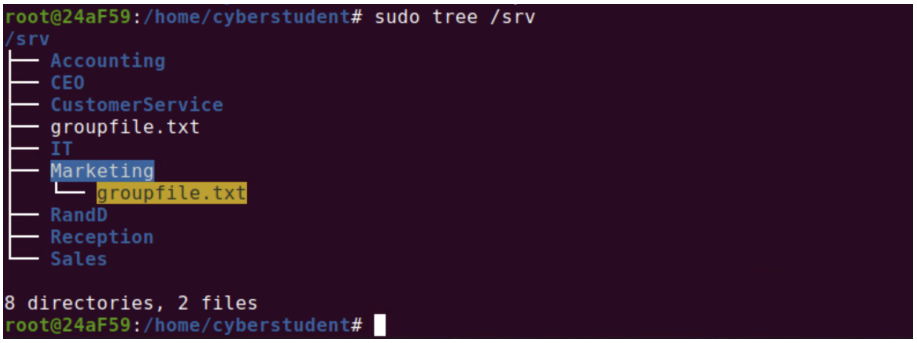

Figure 17: Evidence of Tree Directory

Explanation of Permissions

The permissions set for both the file and directory were read, write, and execute for the owner and group members while others had no permissions. This was to minimize security risks as in the lab manual it was stated that members of the Marketing group should be able to access the files, however, non-administrative users should not. As shown in Figure 20, the octal code 2770 translates to the permissions RWXRWS (Linux Commands - GeeksforGeeks, 2019). In this context, R stands for read, W stands for write, X stands for execute, and S represents the Setgid permission. Figure 7 was used to explain that the octal value 7 grants all permissions—read, write, and execute—applicable to both the owner and the group. Conversely, the octal value 0 indicates no permissions—no read, write, or execute access—and when used as the third digit, it applies to other users (tcarriga, 2020).

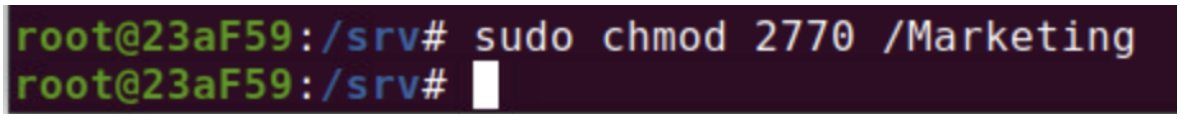

Figure 18: Evidence of Adding Permissions to Folder

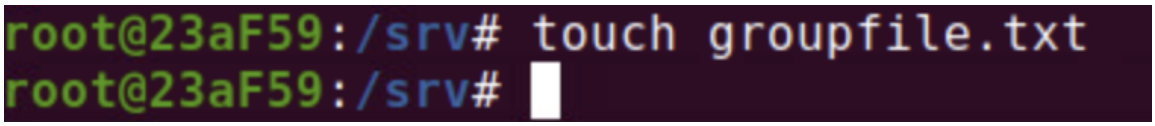

Figure 19: Evidence of File Created

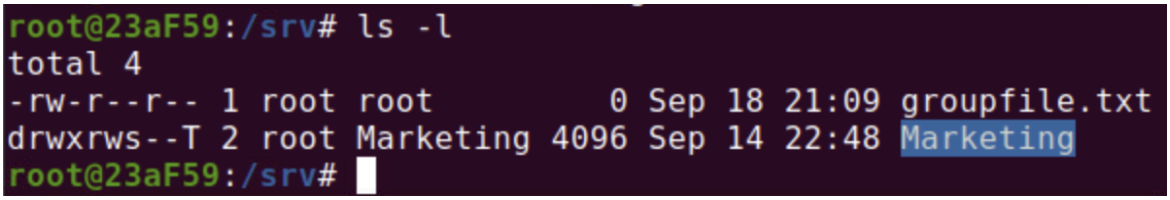

Figure 20: Evidence of Files Shown in List

Permission Creation Methods

The permissions set for the directory was `sudo kd 2770 +t /srv/Marketing` which allowed only the owner and group members of the file to read write and execute as shown in Figure 21. The 2 within the 2770 permission adds a Setgid to the file which makes it so that files and subdirectories created within the directory are automatically owned by the group. Furthermore, the `-t` allowed for a sticky bit to be added that ensured that the creator of a file was the only individual who could delete it as well. This ensured that other users did not delete important files. The permissions that were set to the file were similar as the command that was used was `sudo chmod 2770 /srv/Marketing/groupfile.txt` which sets the file to only be accessed fully by the owner and other group members while people outside the group cannot access it as shown in Figure 23 (Linux Commands - GeeksforGeeks, 2019). These permissions ensure that no non-administrative user can access these files.

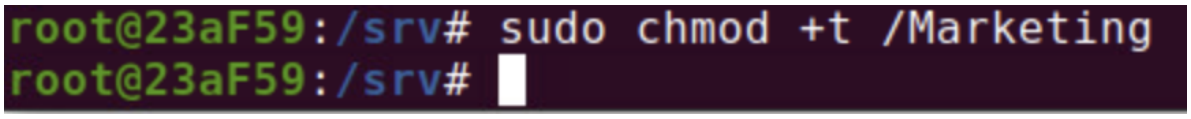

Figure 21: Evidence of Adding Sticky Bit to Marketing Folder

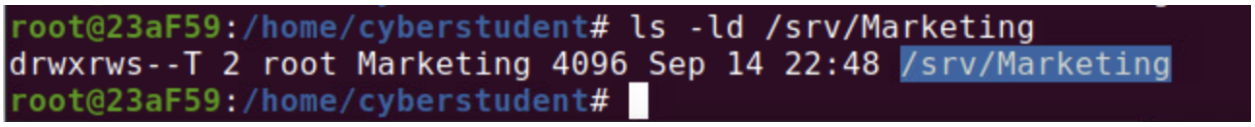

Figure 22: Evidence of Setting Sticky Bit to File

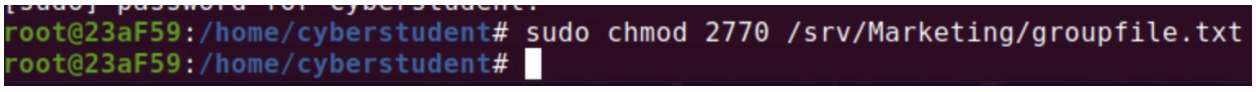

Figure 23: Evidence of Setting Permissions for File

Group Audit

Table 2: Use Cases and Misuse Cases for Shared Files and Folders

General Description	**Command**	**The effect when Authorized User Acts**	**The effect when Unauthorized User Acts**
Create a folder	mkdir [foldername]	Created a new folder (Figure 25)	Permission Denied (Figure 41)
Create a file	touch [filename]	Created a new folder (Figure 27)	Permission Denied (Figure 42)
Modify a folder by Renaming	mv [foldername] [newname]	A folder was renamed (Figure 29)	Permission Denied (Figure 45)
Modify a file	nano [filename]	Opened up a menu where the user can edit the file (Figure 30)	Permission Denied (Figure 46)
Remove folder	rmdir [foldername]	Removed a folder (Figure 32)	Permission Denied (Figure 43)
Remove file	rm [filename]	Removed a file (Figure 34)	Permission Denied (Figure 44)
Misuse Case: Modify a folder another user owns	nano [foldername]	An error popped up and was unable to modify folder (Figure 35)	N/A
Misuse Case: Modify a file another user owns	nano [filename]	An error popped up and was unable to modify file (Figure 36)	N/A
Misuse Case: Alice remove Bob file	rmdir [foldername]	Alice deleted Bob's folder (Figure 38)	N/A
Misuse Case: Alice remove Bob folder	rm [filename]	Alice deleted Bob's file (Figure 40)	N/A

Group Testing Method Authorized/Unauthorized Use Case

The authorized user testing the use case method was Alice. Before testing the methods, the `sudo su Alice` command was used to go into Alice's user account . The `sudo` command allowed Alice to be a superuser (Purdue University. 2016). `Su` stands for substitute user which allowed the switch of users to happen (Purdue University. 2016). Then, `Alice` specifies the account being switched into. Next the `cd /shared/IT_shared` command was used to change the current working directory. The `cd` stands for change directory and helped get from the current directory to the new directory (Purdue University. 2016). The `/shared` refered to the shared root directory (Purdue University. 2016). Then, `IT_shared` is the name of the subdirectory and this one was chosen as Alice is in the IT department (Purdue University. 2016). Throughout some of these methods, the `ls -l` command was utilized to list the files and directories Alice created, removed, or modified. The `-l` specifically stands for long listing format (Purdue University. 2016). Now that Alice was the user and was in the IT_shared directory, an authorized user was able to test the use case methods.

Method 1: The first use case method used was Alice creating a folder using the `mkdir` command (Hands, N. 2024). Since Alice has writing permissions, she was able to create a folder named test_folder (Cipriani, 2020).
Evidence:

```
alice@24aF59:/shared/IT_shared$ mkdir test_folder
alice@24aF59:/shared/IT_shared$
```

Figure 24: Authorized Use Case Method using `mkdir` command

```
alice@24aF59:/shared/IT_shared$ ls -l
total 4
drwxrwsr-x 2 alice IT 4096 Oct  3 12:18 test_folder
alice@24aF59:/shared/IT_shared$
```

Figure 25: Verification Alice's Folder was Created

Method 2: The second use case method used was Alice creating a file in IT_shared directory using the `touch` command (Hands, N. 2024). Since Alice has writing and executing permissions, she was able to create a file named test_file (Cipriani, 2020).
Evidence:

```
alice@24aF59:/shared/IT_shared$ touch test_file
alice@24aF59:/shared/IT_shared$
```

Figure 26: Authorized Use Case Method using `touch` command

```
alice@24aF59:/shared/IT_shared$ ls -l
total 4
-rw-rw-r-- 1 alice IT    0 Oct  3 12:46 test_file
drwxrwsr-x 2 alice IT 4096 Oct  3 12:18 test_folder
alice@24aF59:/shared/IT_shared$
```

Figure 27: Verification Alice's File was Created

Method 3: The third use case method used was Alice renaming a folder in IT_shared directory using the `mv test_folder IT_folder` command (Hands, N. 2024). Since Alice has write and execute permissions, she was able to modify the folder (Cipriani, 2020).
Evidence:

```
alice@24aF59:/shared/IT_shared$ mv test_folder IT_fold
er
alice@24aF59:/shared/IT_shared$
```

Figure 28: Authorized Use Case Method using `mv` command

```
alice@24aF59:/shared/IT_shared$ ls -l
total 8
drwxrwsr-x 2 alice IT 4096 Oct  3 13:29 IT_folder
-rw-rw-r-- 1 alice IT   52 Oct  3 14:14 test_file
alice@24aF59:/shared/IT_shared$
```

Figure 29: Verification Alice's Folder was Renamed

Method 4: The fourth use case method used was Alice modifying a file in IT_shared directory using the `nano test_file` command. The `nano` command allows a user to create, edit or save text files (Hands, N. 2024). With this command Alice was able to open the file and write in it. Since Alice has write and execute permissions, she was able to modify the file (Cipriani, 2020).
Evidence:

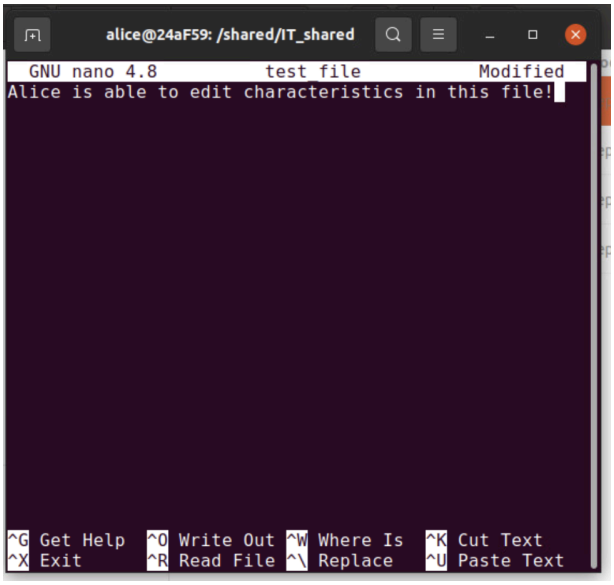

Figure 30: Authorized Use Case Method using `nano` command

Method 5: The fifth use case method used was Alice removing a folder in IT_shared directory using the `rmdir` command (Hands, N. 2024). Since Alice has writing and executing permissions on the parent directory and on the folder, she was able to remove the test_folder (Cipriani, 2020).
Evidence:

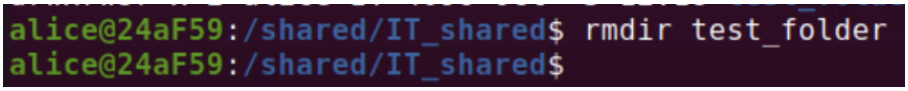

Figure 31: Authorized Use Case Method using `rmdir` command

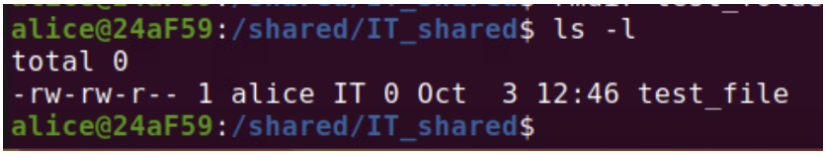

Figure 32: Verification Alice's Folder was Removed

Method 6: The sixth use case method used was Alice removing a file in IT_shared directory using the `rm` command (Hands, N. 2024).Since Alice has writing and executing permissions on the parent directory and on the file, she was able to remove the file (Cipriani, 2020).
Evidence:

Figure 33: Authorized Use Case Method using `rm` command

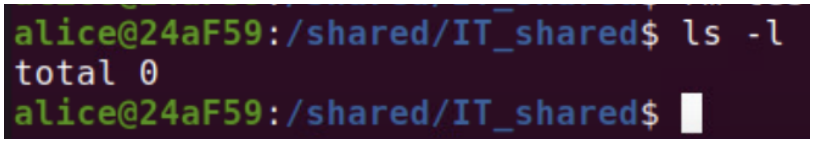

Figure 34: Verification Alice's File was Removed

Group Testing Method Authorized Misuse Case

Before testing the misuse methods, a folder and file was created in Bob's account so there were folders and files in the IT_shared directory that was created by someone else besides Alice. To do this, first the `sudo su` command was used to get into Bob's account. Next, the `cd /shared/IT_shared` command was used since Bob was also in the IT department. Then, the `mkdir` and `touch` command was used so he could create a folder named bob_folder and a file named bob_file. Last step was to switch accounts back to Alice's account.

Method 1: The first misuse case method used was Alice modifying a folder another user owns. The `nano` command was used again, except for bob's folder instead of Alice's. Even though both Alice and Bob are in the IT department, Alice was unable to write in Bob's folder because

Alice has read-only permissions. This means she cannot modify Bob's folder. As a result, an error popped up saying permission denied when trying to save the modification.
Evidence:

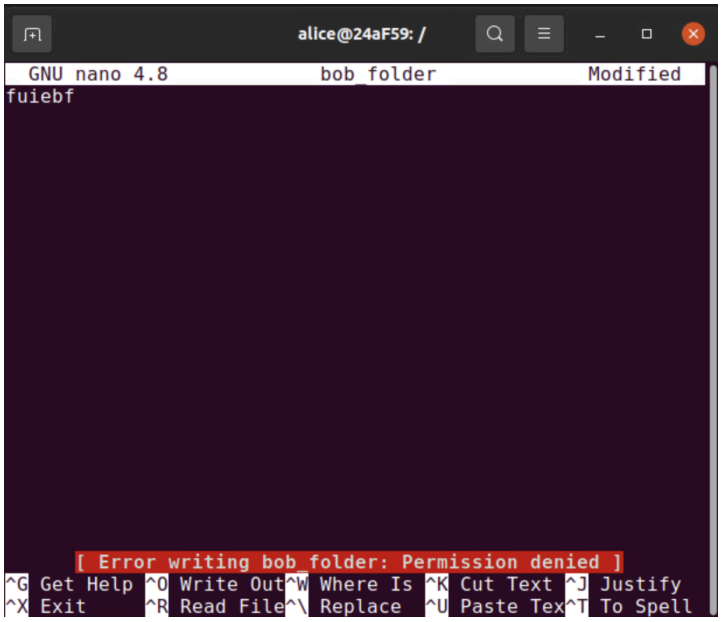

Figure 35: Authorized Misuse Case Method using `nano [foldername]` command

Method 2: The second misuse case method used was Alice modifying a file another user owns. The `nano` command was used again, except for Bob's file instead of Alice's. Even though both Alice and Bob are in the IT department, Alice was unable to write in Bob's file because Alice has read-only permissions. This means she cannot modify Bob's file As a result, an error popped up saying permission denied when trying to save the modification.

Evidence:

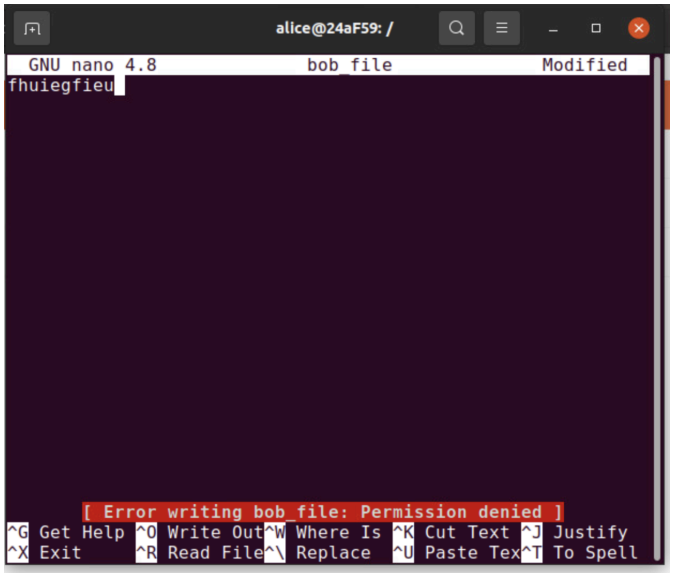

Figure 36: Authorized Misuse Case Method using `nano [filename]` command

Method 3: The third misuse case was Alice removing a folder that she did not create. This was done using the `rmdir` command. Since Alice has administrative permissions (read, write, execute), she was able to delete bob's folder.
Evidence:

```
alice@24aF59:/shared/IT_shared$ rmdir bob_folder
alice@24aF59:/shared/IT_shared$
```

Figure 37: Authorized Misuse Case Method using `rmdir` command

```
alice@24aF59:/shared/IT_shared$ ls -l
total 8
-rw-rw-r-- 1 bob   IT    0 Oct  3 19:26 bob_file
drwxrwsr-x 2 alice IT 4096 Oct  3 13:29 IT_folder
-rw-rw-r-- 1 alice IT   52 Oct  3 14:14 test_file
alice@24aF59:/shared/IT_shared$
```

Figure 38: Verification Alice was able to delete Bob's folder

Method 4: The fourth misuse case was Alice removing a file that she did not create. This was done using the `rm` command. Since Alice has administrative permissions (read, write, execute), she was able to delete Bob's file.
Evidence:

```
alice@24aF59:/shared/IT_shared$ rm bob_file
alice@24aF59:/shared/IT_shared$
```

Figure 39: Authorized Misuse Case Method using `rm` command

```
alice@24aF59:/shared/IT_shared$ ls -l
total 8
drwxrwsr-x 2 alice IT 4096 Oct  3 13:29 IT_folder
-rw-rw-r-- 1 alice IT   52 Oct  3 14:14 test_file
alice@24aF59:/shared/IT_shared$
```

Figure 40: Verification Alice was able to delete Bob's file

Group Testing Method Unauthorized Misuse Case

The unauthorized user was Carol because she is not in the IT department, therefore, she has different permissions from Alice and Bob. First, the `sudo su` command was utilized to get into her account. Then, each command that was done for Alice was repeated.

Method 1: The first method was Carol creating a folder. Carol received an error saying permission denied as she does not have the write permissions, therefore, she cannot create a folder.

Evidence:

```
carol@24aF59:/shared/IT_shared$ mkdir carol_folder
mkdir: cannot create directory ‘carol_folder’: Permission denied
carol@24aF59:/shared/IT_shared$
```

Figure 41: Unauthorized Misuse Case Method using `mkdir` command

Method 2: The second method was Carol creating a file. Carol received an error saying permission denied as she does not have the write permissions, therefore, she cannot create a file.
Evidence:

```
carol@24aF59:/shared/IT_shared$ touch carol_file
touch: cannot touch 'carol_file': Permission denied
carol@24aF59:/shared/IT_shared$
```

Figure 42: Unauthorized Misuse Case Method using `touch` command

Method 3: The third method was Carol deleting IT_folder. Carol received an error saying permission denied as she does not have the write or execute permissions, therefore, she cannot delete a folder.
Evidence:

```
carol@24aF59:/shared/IT_shared$ rmdir IT_folder
rmdir: failed to remove 'IT_folder': Permission denied
carol@24aF59:/shared/IT_shared$
```

Figure 43: Unauthorized Misuse Case Method using `rmdir` command

Method 4: The fourth method was Carol deleting test_file. Carol received an error saying permission denied as she does not have the write or execute permissions, therefore, she cannot delete a file.
Evidence:

```
carol@24aF59:/shared/IT_shared$ rm test_file
rm: cannot remove 'test_file': Permission denied
carol@24aF59:/shared/IT_shared$
```

Figure 44: Unauthorized Misuse Case Method using `rm` command

Method 5: The fifth method was Carol modifying IT_folder. Carol received an error saying permission denied as she does not have the write or execute permissions, therefore, she cannot modify a folder.
Evidence:

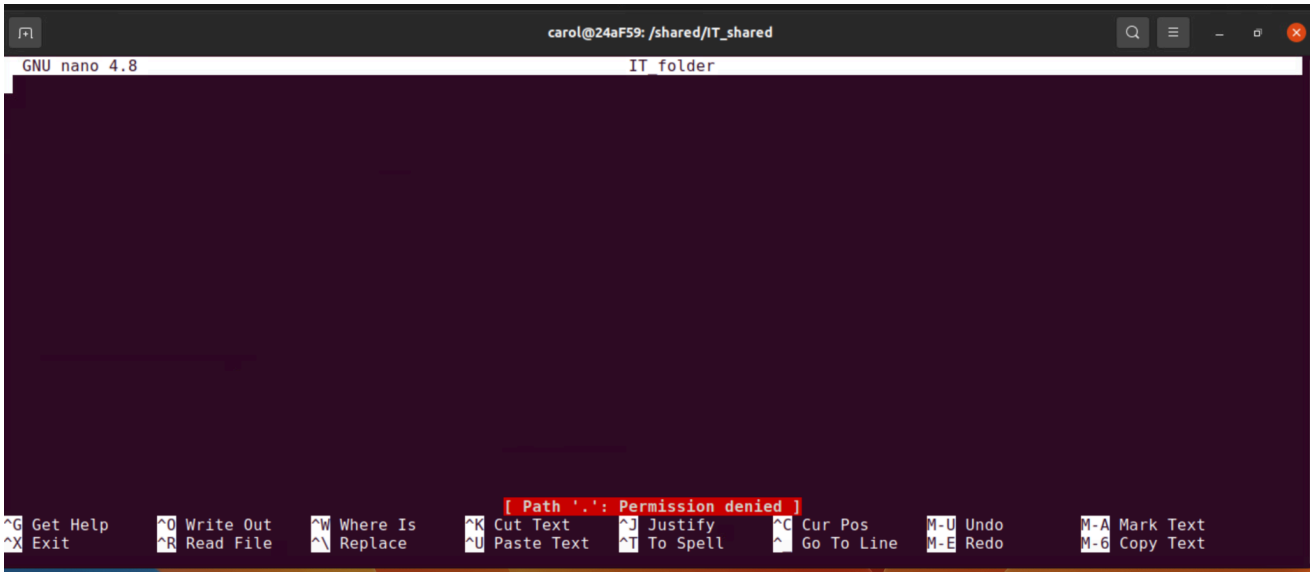

Figure 45: Unauthorized Misuse Case Method using `nano` command on Folder

Method 6: The sixth method was Carol modifying test_file. Carol received an error saying permission denied as she does not have the write or execute permissions, therefore, she cannot modify a file.

Evidence:

Figure 46: Unauthorized Misuse Case Method using `nano` command on File

Root Account

Disable Root Login Method

To disable the root login, edit the `/etc/passwd` file to `sudo nano /etc/passwd`. Then, change the root user's shell from `/bin/bash` to `/sbin/nologin` (Pathak, 2020). Figure 27 displays root L which stands for locked, confirming the root is disabled.

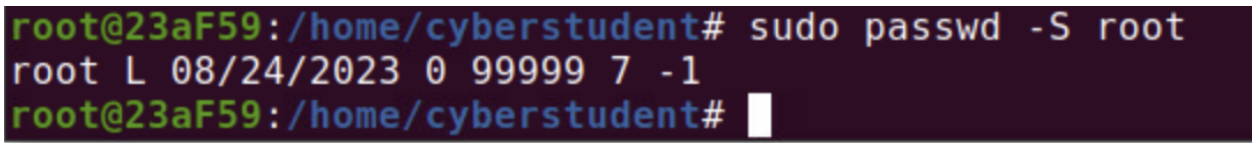

Figure 27: Evidence of Disabled Root Login

Reenable Root Login Method

To re-enable the root login, use the `sudo passwd -u root` command (Pathak, 2020). Figure 20 displays a root P which means, root password, proving the root login is re-enabled.

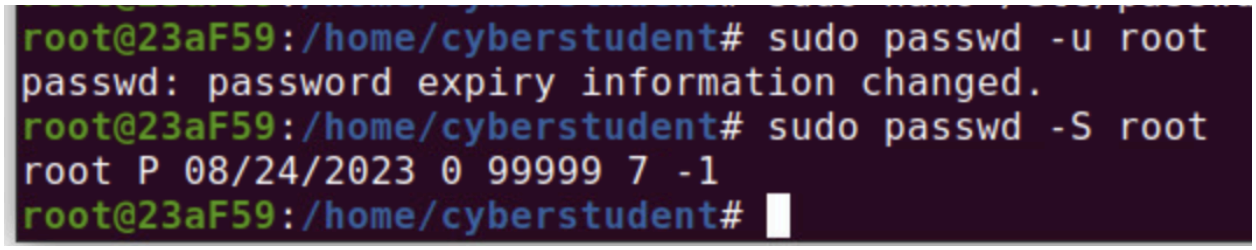

Figure 28: Evidence of Reenabled Root Login

/etc/passwd & /etc/shadow

Explanation of the colon-delimited fields in the /etc/passwd file in plain language
The `/etc/passwd` file contained information about users that all users can read. This information is separated into fields by colons(Understanding the /etc/passwd File - Linuxize,2019). The first field is the name of the user which in the case of this business scenario can be Alice, Bob, and etc. The second field is the password placeholder which is represented by an x which means that this user had a password. The third field is the user ID that identified the user in the system. The fourth field in this is the Group ID which states the ID numbers of the groups that the user is associated with. The fifth field is user info which stores full names, addresses, and etc. The sixth field contains the directory in which the user's files are stored for example Alice's home directory would show /home/alice. Lastly, the seventh and final field shows the command-line interface that runs after the user logs in this case scenario it would be /bin/bash(Understanding the /etc/passwd File - Linuxize,2019).

Explanation of the colon-delimited fields in the /etc/shadow file in plain language
The /etc/shadow file stores sensitive information about users and is only readable by the system administrator or the root(Understanding the /etc/shadow File - Linuxize,2019). Just like the /etc/passwd file each line corresponds to a user however within this file the sensitive information is hashed/encrypted. The first field is the user's name the same as the passwd file. The second field is their password and unlike the passwd file, this one shows the full password but hashed to protect the user's account. The third field is the number of days since the last password change and is stored as the number of days since January 1st, 1970. The fourth field is the number of days till the user can change their password again. The fifth field is the maximum amount of days a user can keep their password without changing it. The sixth field is the number of days before the user gets a warning to change their password. The seventh field is the amount of days the user has been inactive which means the amount of days since their password has expired. The eighth field is the day the password will expire stored in days since January 1st, 1970. Lastly, the ninth field is a special reserved field stored for future use and is also typically empty(Understanding the /etc/shadow File - Linuxize,2019).

Create Character Devices

Explanation for /dev/random having 444 permissions
The octal code 444 according to Figure 7 grants owners, groups, and others the ability to read files but not write or execute them. This is important for the file /dev/random as this device provides randomly generated numbers generated from unpredictable sources used for cryptographic purposes. If a bad actor had the ability to compromise this randomness they can use it to figure out certain sensitive keys and tokens. This can create potential threats within the system that can be exploited (Sharma, 2023).

Explanation for /dev/tty having 666 permissions
The 666 octal code permission according to Figure 7 gives owners, groups, and others the ability to read and write. This is necessary for /dev/tty as this represents terminal access and console processes. It is essential for users to have read and write privileges as users must be able to input commands through writing and read the outputs. Unlike /dev/random, /dev/tty has no security risk from the added write permission as this only affects the user's terminal access and not other individuals or systems.

Chroot Jail: Testing

Note: In this section, screenshots were initially taken without positioning the cursor on the next line, prompting a need to retake them. Since the groups already existed, a message appeared indicating this during the process. As a result, both the original and updated screenshots were included to provide a complete view of the actions taken.

Users Jailed Method- Create a "remoteuser" group and Add a User

A chroot jail is a fake directory that makes the user think they are a part of the real directory but they are not. Since Handy Hardy Hardware decided to move remote, a new group named remote users needed to be created. As shown in Figure 29, the `groupadd [username]` command was used again to create the remote users group. Next, Alice and Bob were testing the setup with Alice and Carol's user accounts, so Alice was assigned root permission. The permission was given through the `usermod -aG remoteusers [username]` command. Figure 30 shows this process for Alice, the same was done for Carol. Then as shown in Figure 32, the tail command listed the users with sudo permission to confirm they were given to Alice and Carol.

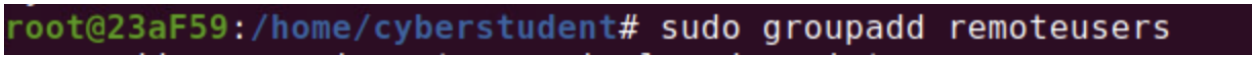

Figure 29: Old Evidence of Remoteusers Created into a Group

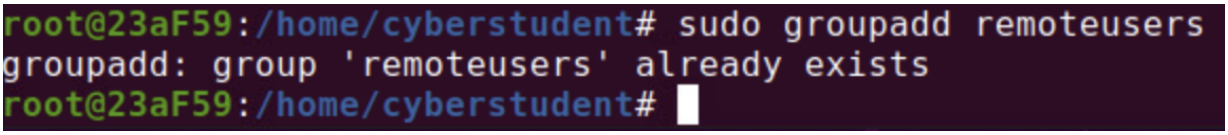

Figure 30: New Evidence of Remoteusers Created into a Group

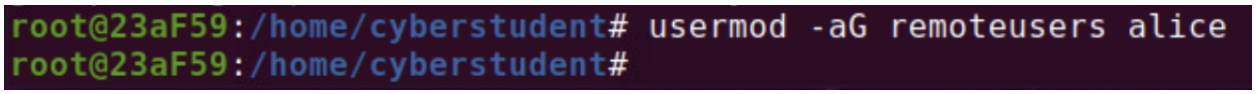

Figure 31: Evidence of Alice Receiving Sudo Permission

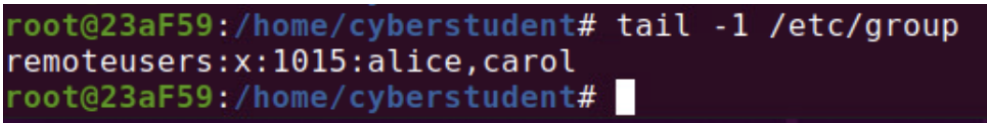

Figure 32: Evidence of Alice and Carol in the Remoteusers Group

Users Jailed Method- Create File Structure & Evidence

To create a new directory, the `sudo mkdir -p detention` command was used to name the new directory 'detention' with administrative privileges inside the chroot jail. Next, subdirectories were created within the main detention directory using the command sudo `mkdir -p /detention/[subdirectory]` (Bash Mkdir and Subfolders, n.d.). Figure 34 shows the usr subdirectory being created in the detention directory and this step was repeated for the subdirectories dev, etc, lib, and bin. Additionally, the command sudo `mkdir -p`

`/detention/usr/bin` was used to create a subdirectory bin within the usr directory inside detention (Purdue University. 2016). Finally, Figure 36 displays the `sudo tree /detention` command to show that the detention directory has the required directory structure.

Figure 33: Old Evidence of Creating a New Directory, Detention

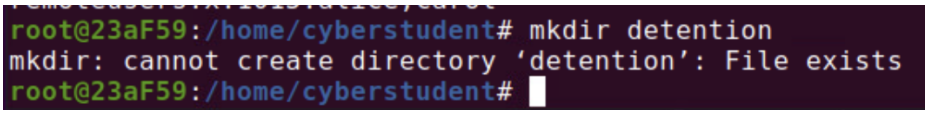

Figure 33: New Evidence of Creating a New Directory, Detention

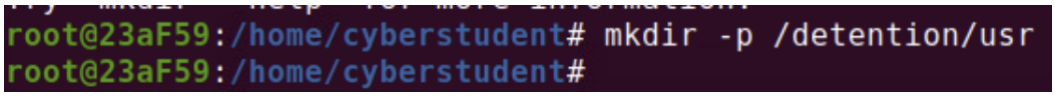

Figure 34: Evidence of Subdirectory Usr in Detention

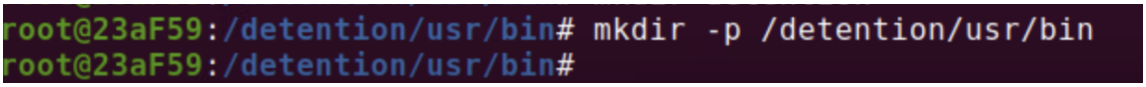

Figure 35: Evidence of Creating a Subdirectory in the Usr Directory

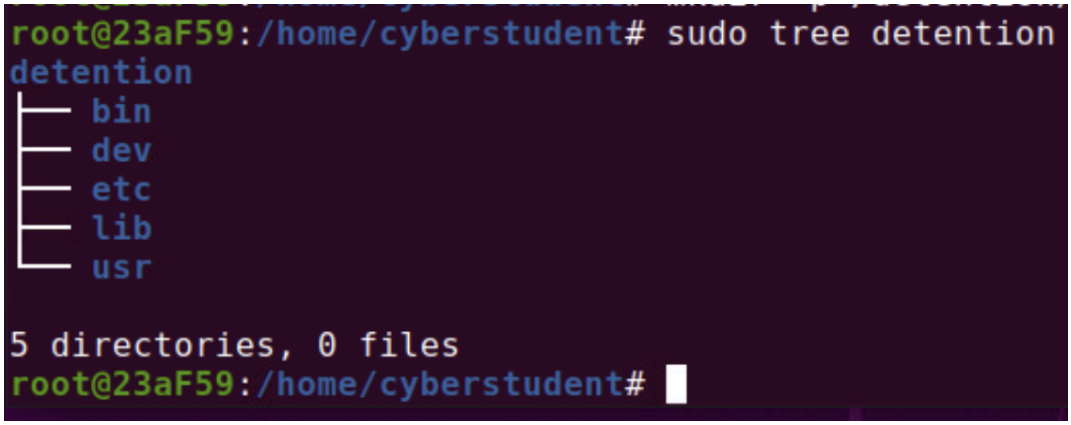

Figure 36: Evidence that detention has the required directory structure using the tree command

Users Jailed Method- Create Character Device

A character device is a type of hardware device that sends and receives data one byte at a time, instead of in blocks. Character devices like /dev/null, /dev/zero, /dev/tty, /dev/random, and /dev/urandom are designed to give basic operations on Linux in situations when the user's root is limited to a folder with an extremely restricted environment lacking many required characteristics (Exploring /Dev/Random vs. /Dev/Urandom and /Dev/Zero vs. /Dev/Null, 2010).

This configuration guarantees that necessary operations can be carried out as planned, just like in a typical root environment. Use the following command to create these devices:

```
mknod -m [permission bits] [path of a file] c [major number]
                          [minor number]
```

The character devices used for this chroot jail are:

```
mknod -m 666 /detention/dev/null c 1 3
mknod -m 622 /detention/dev/zero c 1 5
mknod -m 666 /detention/dev/tty c 5 0
mknod -m 444 /detention/dev/random c 1 8
mknod -m 444 /detention/dev/urandom c 1 9
```

Figure 37 shows an example of the `mknod -m 666 /detention/dev/null c 1 3` character device being created. `Mknod` is the tool that helped create the special files that represent hardware devices. The flag -m indicates the device's authorization setting. Octal bits are used to represent permission bits that tell the system which device is being created. Within the designated path, the path will generate a device. Character devices are the type of device that will be manufactured, as indicated by the letter c. The major and minor numbers are included afterward, with each number separated by space. Additionally, the `chown root:tty /detention/dev/tty` command is used to ensure full functionality of /detention/dev/tty.

Figure 37: Old Evidence of Creating the Character Devices

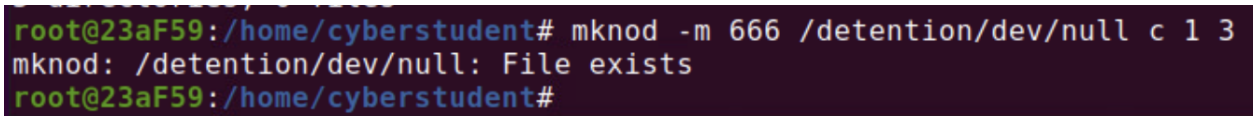

Figure 38: New Evidence of Creating the Character Devices

Users Jailed Method-Copy files from the regular filesystem

To ensure other basic functionalities are provided to users in the chroot jail, non-character device files that are necessary for chroot jail to function are copied from root to /detention using `/cd and cp [from path] [target path].` The ‘ .’ at the end of each command in Figures 40-43 signifies the file is being copied. Figures 44-46 are additional files that were copied from `/bin/ls` and `/bin/bash` to `/detention/bin`.

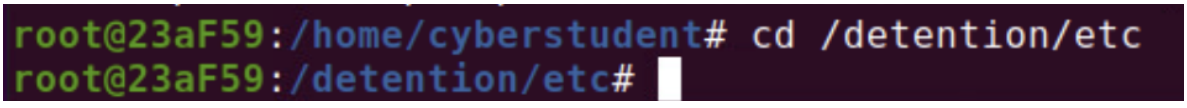

Figure 39: Evidence of Changing the Current Directory

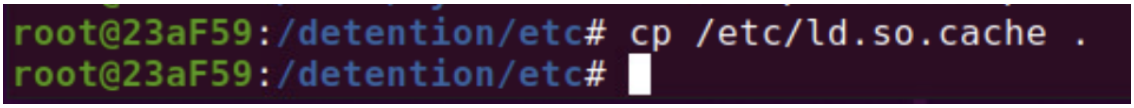

Figure 40: Evidence of Copying the ld.so.cache File to the /etc Directory

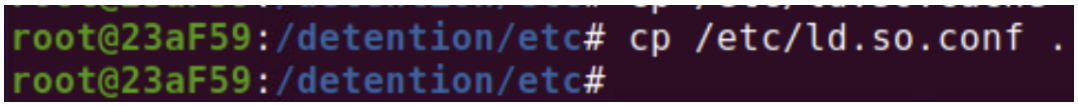

Figure 41: Evidence of Copying the ld.so.conf File to the /etc Directory

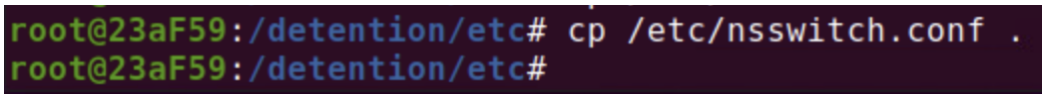

Figure 42: Evidence of Copying the nsswitch.conf File to the /etc Directory

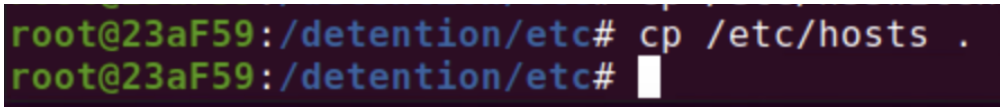

Figure 43: Evidence of Copying the Host's File to the /etc Directory

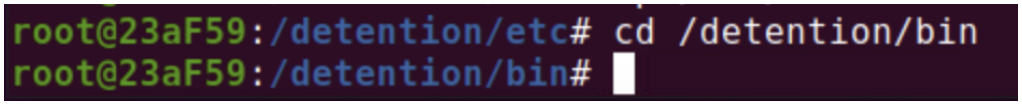

Figure 44: Evidence of Switching to Binary Directory

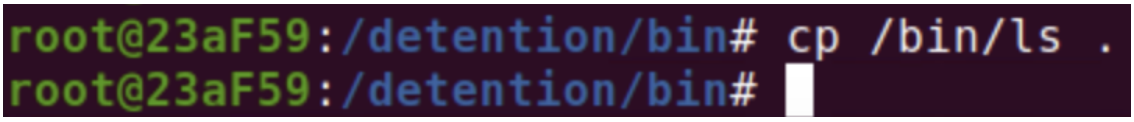

Figure 45: Evidence of Copying the Binaries into the Chroot Jail

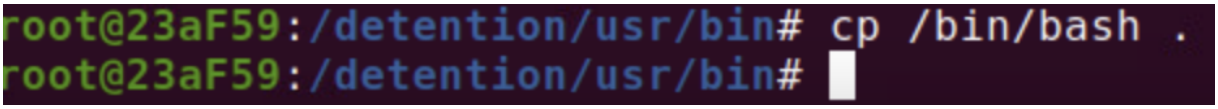

Figure 46: Evidence of Executing the Chroot Jail

The `ldd` command was used to identify shared libraries that were used by the `ls` command.

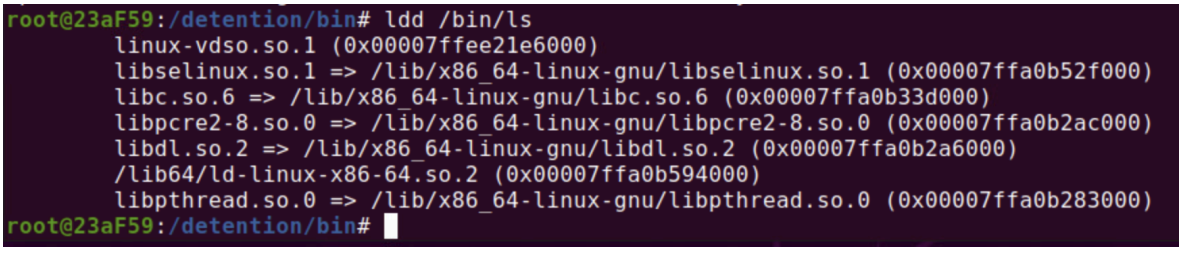

Figure 47: Evidence of Creating a Shared Library

```
root@23aF59:/detention/bin# cd /detention/usr/bin
root@23aF59:/detention/usr/bin#
```

Figure 48: Evidence of Changing Directories

After identifying the list of shared libraries used by ls command, a script is used to automatically copy necessary files. The script is copied from `/home/cyberstudent/l2chroot to /detention/usr/bin.`

```
root@23aF59:/detention/usr/bin# sudo cp /home/cyberstudent/l2chroot .
root@23aF59:/detention/usr/bin#
```

Figure 49: Evidence of Copying the l2chroot File

Next, the 755 permissions were set to l2chroot. The 7 means the admin can read, write, and execute. The 5 means read and write permissions only, so that was given to the groups and others.

```
root@23aF59:/detention/usr/bin# sudo chmod 755 l2chroot
root@23aF59:/detention/usr/bin#
```

Figure 50: Evidence of Setting Permissions

```
root@23aF59:/detention/usr/bin# sudo nano l2chroot
root@23aF59:/detention/usr/bin#
```

Figure 51: Evidence of Modifying the File to Chroot Directory

Users Jailed Method- Using nano to alter files

Following the copy of the script file, the script file is further modified to alter the base of the folder for chroot jail from /jailed to /detention. Once the file was saved, exited out of nano using ctrl + o, enter, then ctrl + x.

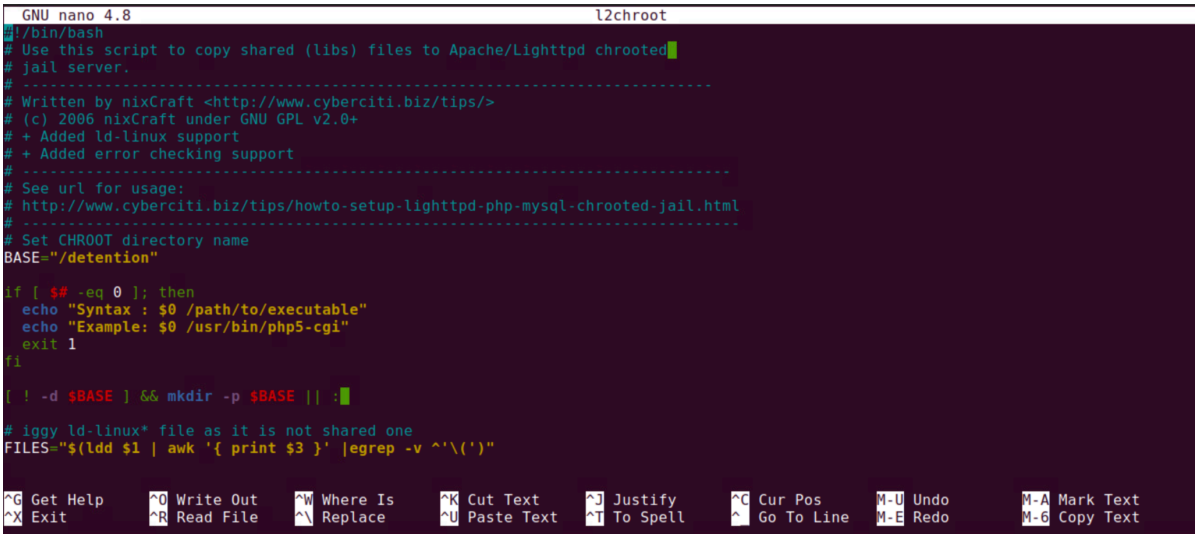

Figure 52: Evidence of Altering file of l2chroot

```
root@23aF59:/detention/usr/bin# sudo /detention/usr/bin/l2chroot /bin/ls
Copying shared files/libs to /detention...
root@23aF59:/detention/usr/bin#
```

Figure 53: Evidence of Script operation for `ls` program

```
root@23aF59:/detention/usr/bin# sudo /detention/usr/bin/l2chroot /bin/bash
Copying shared files/libs to /detention...
root@23aF59:/detention/usr/bin#
```

Figure 54: Evidence of Script operation for `bash` program

```
root@23aF59:/detention/usr/bin# nano /etc/ssh/sshd_config
root@23aF59:/detention/usr/bin#
```

Figure 55: Evidence of Opening File to Edit

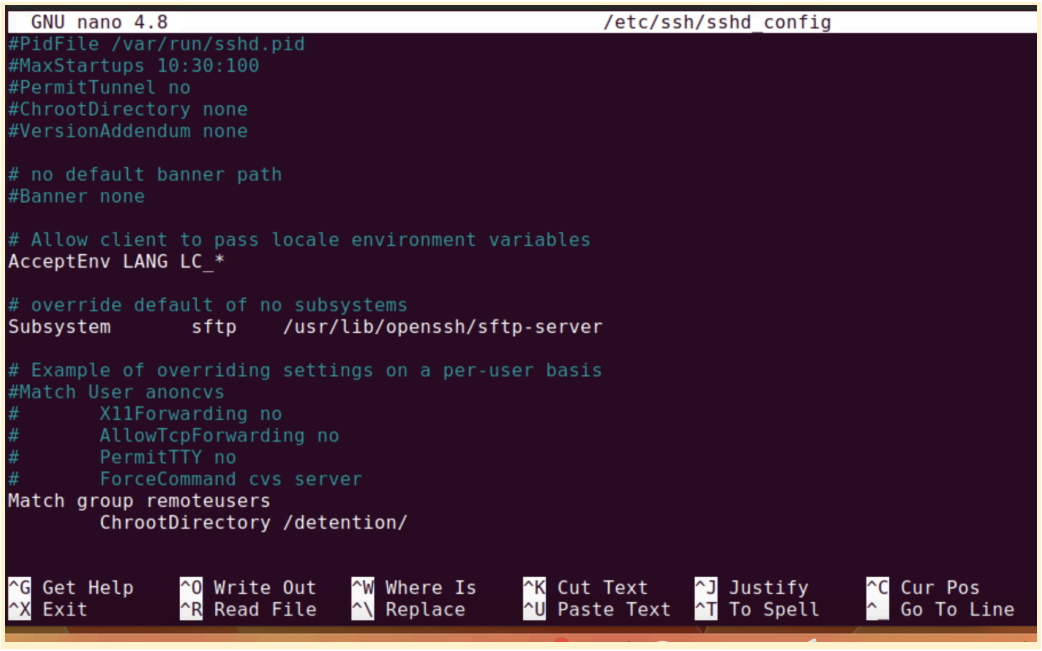

Figure 55: Evidence of Configuration of SSH Daemon

Chroot Jail Testing

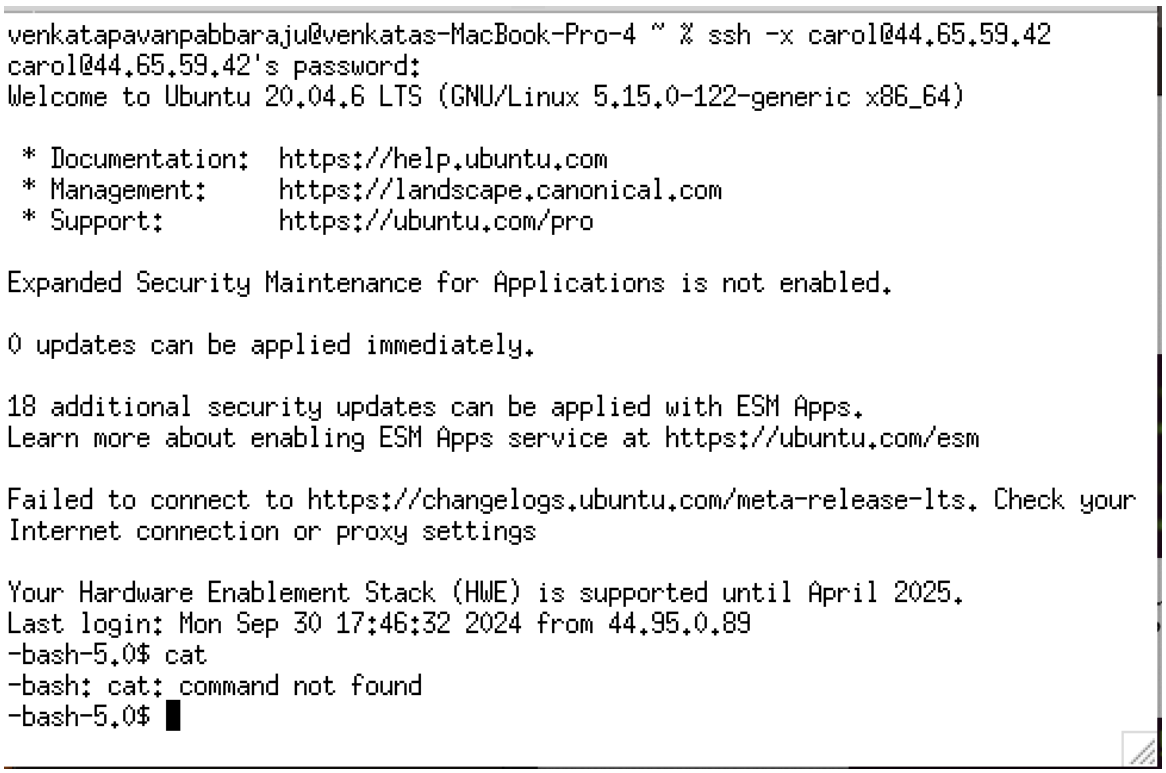

Figure 56: Evidence Chroot Jail Works

The chroot jail was tested on VMware Server B, requiring an IP address to connect via SSH. To locate the IP address, the `host -i` command was used. Once identified, the SSH connection

was initiated using `[username]@[IP Address]password` to access the Ubuntu system. However, when attempting to execute commands, the system returned an error stating "command not found," indicating that the chroot jail was functioning correctly by restricting access. As a result, both Alice and Carol were successfully prevented from accessing restricted directories, demonstrating that the chroot jail was effective in confining users within the specified environment.

Extra Credit:

The chroot jail limits what files the directories can access. So, the "I have no name" error appears in a chroot environment when the system cannot find the current user's ID to a username because the `/etc/passwd` file is incorrectly configured (Gite, 2017). To fix this, `sudo cp /etc/passwd /detention/etc` copied the `/etc/passwd` file into the `chroot directory, /detention/etc`. Now, when Alice is asked "whoami", the system can respond with 'alice'. Additionally, this response confirms the Chroot jail is working and configured correctly as it can find userID without any errors.

```
alice@23aF59:~$ sudo cp /etc/passwd /detention/etc/
alice@23aF59:~$
```

Figure 57: Evidence of Copying the File into the Directory

```
alice@23aF59:~$ whoami
alice
alice@23aF59:~$
```

Figure 58: Evidence of Correct User Logged-In

Chroot Jail: Making the Jail Nice

Mounting Home and File Share Methods

To mount home and shared folders in a chroot prison, the procedure involves replicating the real root's (/) fundamental file structure within the chroot environment. As part of this procedure, directories like /home and /shared are created inside the chroot, and bind mounts—such as mount --bind /real/home /chroot/home—are used to connect these chroot directories to their real counterparts (Mount Command in Linux with Examples, 2019). Through this configuration, users inside the chroot prison can access their home and shared resources while remaining disconnected from the real filesystem, thus simulating the true root structure in a secure and controlled environment.

The `mount` command enables the filesystem to attach itself to the large tree structure. –bind flag indicates the type of mount, where the mount for bind will be to replicate a directory that already exists on the big tree to a different directory (Mount Command in Linux with Examples, 2019). Figures 64 and 65 showcase both the shared folders on the regular and detention directory as the same. Figures 66 and 67 present the home folder of the regular directory and the detention directory being the same.

Creates Mount Points Evidence

```
root@23aF59:/detention# mkdir home
mkdir: cannot create directory ‘home’: File exists
root@23aF59:/detention#
```

Figure 59: Evidence of Creating Home Directory in /detention

```
root@23aF59:/detention# mkdir sharedFolder
mkdir: cannot create directory ‘sharedFolder’: File exists
root@23aF59:/detention#
```

Figure 60: Evidence of Creating shared folder directory in /detention

```
root@23aF59:/detention# ls -a
.  ..  bin  dev  echo_test  etc  hom  home  lib  lib64  share  sharedFolder  usr
root@23aF59:/detention#
```

Figure 61: Evidence of Creating folders within /detention directory

Bind Mount Evidence

```
root@23aF59:/detention# mount --bind /home /detention/home
root@23aF59:/detention#
```

Figure 62: Evidence of Creating Mount of /home to /detention/home

```
root@23aF59:/sharedFolder# mount --bind /sharedFolder /detention/sharedFolder
root@23aF59:/sharedFolder#
```

Figure 63: Evidence of Creating Mount of /sharedFolder to /detention/sharedFolder

```
cyberstudent@23aF59:/sharedFolder$ ls -a
.  ..  accounting  Execs  HR  it  Sales
cyberstudent@23aF59:/sharedFolder$
```

Figure 64: Evidence of Creating a List of Directories in /sharedFolders

```
root@23aF59:/detention/sharedFolder# ls -a
.  ..  accounting  Execs  HR  it  Sales
root@23aF59:/detention/sharedFolder#
```

Figure 65: Evidence of Creating a List of Directories in /detention /shared folders

```
root@23aF59:/home# ls -l
total 40
drwx------  5 alice       alice       4096 Sep 17 13:50 alice
drwxr-xr-x  2 bob         bob         4096 Sep 10 13:08 bob
drwx------  2 carol       carol       4096 Sep 14 22:47 carol
drwxr-xr-x 20 cyberstudent cyberstudent 4096 Sep 17 12:43 cyberstudent
drwxr-xr-x  2 eve         eve         4096 Sep 10 13:10 eve
drwxr-xr-x  2 hands       hands       4096 Sep 17 13:09 hands
drwxr-xr-x  2 malory      malory      4096 Sep 10 13:11 malory
drwxr-xr-x  2 trent       trent       4096 Sep 10 13:12 trent
drwxr-xr-x  2 victor      victor      4096 Sep 10 13:16 victor
drwxr-xr-x  2 walter      walter      4096 Sep 10 13:15 walter
root@23aF59:/home#
```

Figure 66: Evidence of Creating List of users in /home

```
cyberstudent@23aF59:/detention/home$ ls -l
total 40
drwx------  5 alice       alice       4096 Sep 17 13:50 alice
drwxr-xr-x  2 bob         bob         4096 Sep 10 13:08 bob
drwx------  2 carol       carol       4096 Sep 14 22:47 carol
drwxr-xr-x 20 cyberstudent cyberstudent 4096 Sep 17 12:43 cyberstudent
drwxr-xr-x  2 eve         eve         4096 Sep 10 13:10 eve
drwxr-xr-x  2 hands       hands       4096 Sep 17 13:09 hands
drwxr-xr-x  2 malory      malory      4096 Sep 10 13:11 malory
drwxr-xr-x  2 trent       trent       4096 Sep 10 13:12 trent
drwxr-xr-x  2 victor      victor      4096 Sep 10 13:16 victor
drwxr-xr-x  2 walter      walter      4096 Sep 10 13:15 walter
cyberstudent@23aF59:/detention/home$
```

Figure 67: Evidence of Creating List of users in /detention /home

Binary & Shared Libraries Method

```
root@23aF59:/detention/bin# cp /bin/vim.tiny .
root@23aF59:/detention/bin#
```

Figure 68: Evidence of cp command to copy vim.tiny to /detention/bin

```
root@23aF59:/detention/bin# cp /bin/echo .
root@23aF59:/detention/bin#
```

Figure 69: Evidence of use of cp command to copy echo to /detention/bin

```
root@23aF59:/detention/bin# cp /bin/tail .
root@23aF59:/detention/bin#
```

Figure 70: Evidence of use of cp command to copy tail to /detention/bin

Binary & Shared Libraries Method

To provide more functionality for chroot jail users, a few more binaries had to relocate once the shared folders between chroot jail and real root were moved. Tiny vim, tail, and echo are among the binaries that had to move. This was done by copying the binaries from our real root and giving them to the detention folder through the `cp` command shown in Figures 68, 69, and 70. The `cp` command copies files and directories from one location to another (Hands, N. 2024). This allows for the chroot jail to have the same abilities and access to modify files and folders as the real root.

```
root@23aF59:/detention/usr/bin# /detention/usr/bin/l2chroot /bin/vim.tiny
Copying shared files/libs to /detention...
root@23aF59:/detention/usr/bin#
```

Figure 71: Evidence of l2chroot copying shared libraries

Binary Testing Method

To verify that the binaries inside the chroot prison work, each of them was executed after gaining access to the chroot environment. This is shown in Figure 73 when vim.tiny is started, it verified that the text editor is operating correctly by responding with the number of files to edit. If there are no issues during these types of tests, then the necessary libraries and dependencies are appropriately included in the chroot environment. The binaries are operating as expected, proving the chroot environment is configured appropriately for jailed users. Proper command execution, accurate output, and successful file operations all contribute to this conclusion.

Binary Functionality Evidence

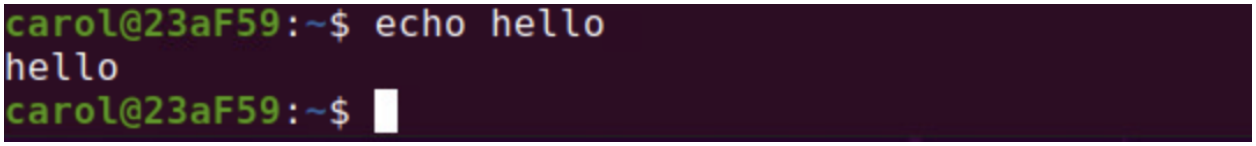

Figure 72: Evidence of use of echo function

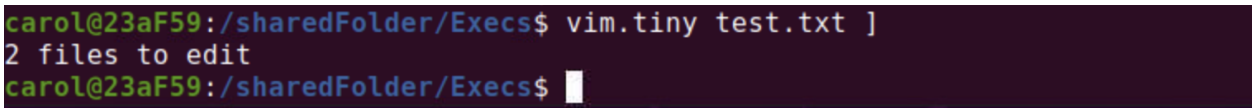

Figure 73: Evidence of use of vim.tiny function

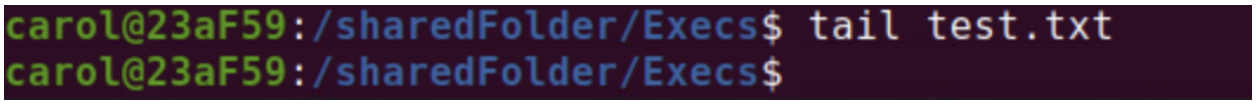

Figure 74: Evidence of use of tail function

Chroot Jail: Nice Jail Testing

Share Mount Testing Method/Evidence

A share mount testing method refers to the process of assessing the capacity to mount and access a shared directory or resource across a network To be able to test share mount, an additional file or directory is created within each of the two mounted directories using one of two platforms: virtual machines or SSH. This will guarantee that any changes made to the true root or /detention directories will also affect the other location. An example of this is shown when the shared folder in Figure 76 is identical to the detention folder in Figure 78. Both folders display is "`-rw-r-r-.`" This means there is functionality within the chroot jail. Similar to how the regular directory is based on the user's profiles and allows for the users to be able to access their folders without posing any threat to the real Directory.

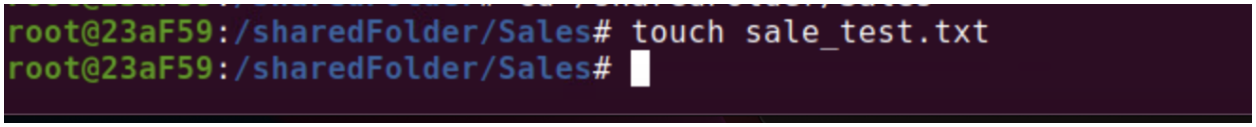

Figure 75: Creating text file in Real Directory

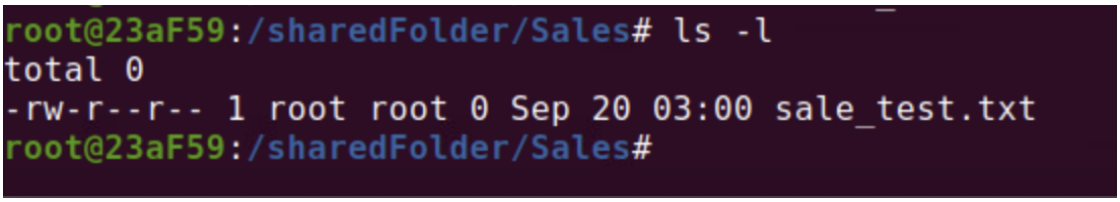

Figure 76: Evidence of Permissions in /sharedFolder/sales

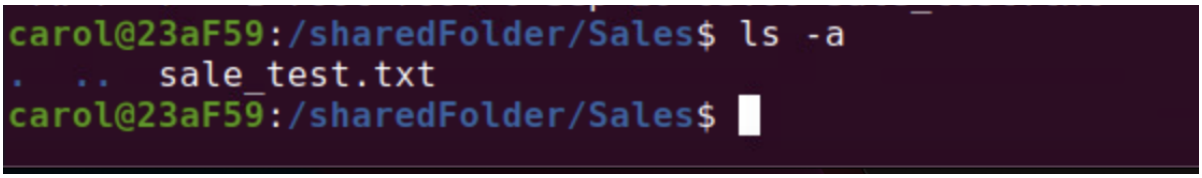

Figure 77: Files in Chroot Jail

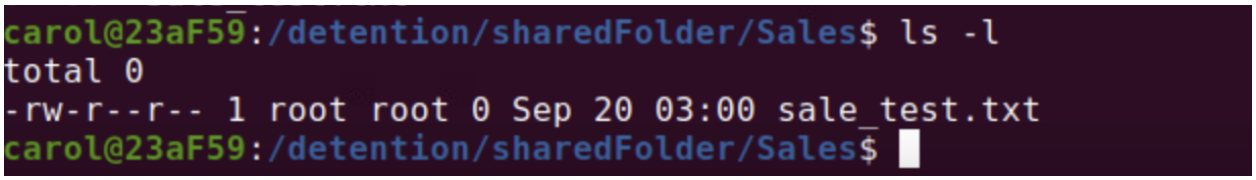

Figure 78: Details of Files in Chroot Jail

After the shared file was created, the user Carol connected to the system via SSH. Carol's account is configured to operate within a chroot jail, meaning that her environment is isolated from the rest of the filesystem. In this setup, the shared folder is located at /detention/sharedFolder. The ls -l command was used in Figure 86 to check the file permissions.

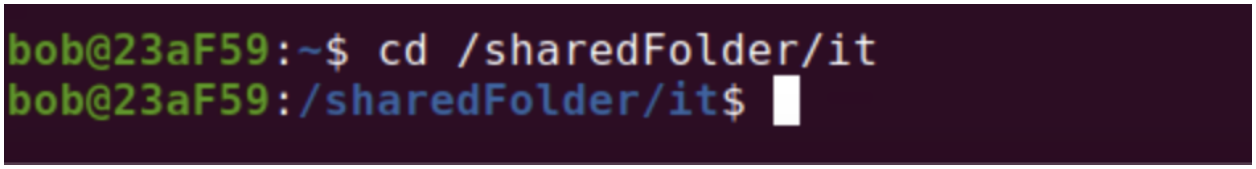

Figure 79: Evidence of Change Directory to bob

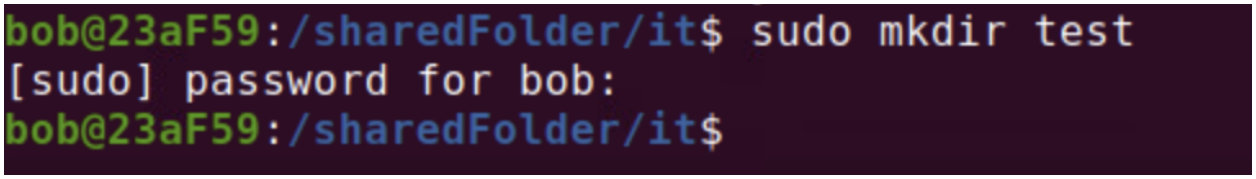

Figure 80: Evidence of Creating Directory Test for bob

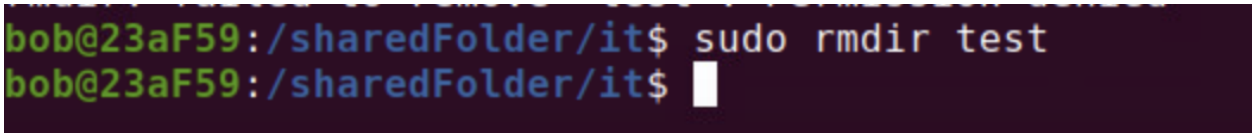

Figure 81: Evidence of Removing Directory Test for bob

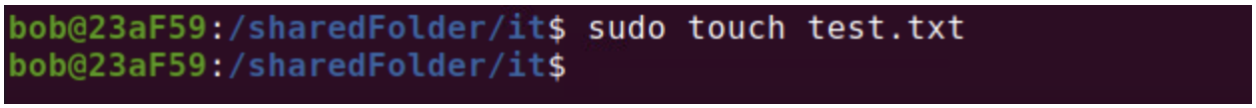

Figure 81: Evidence of Creating test.txt File for bob

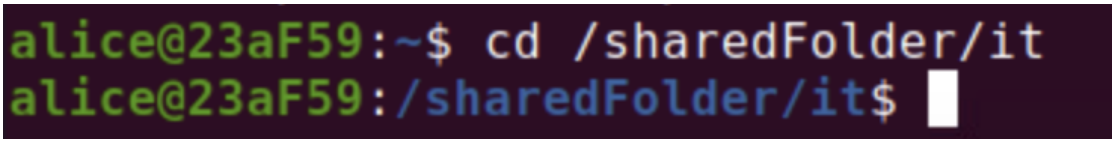

Figure 82: Evidence of Change Directory to alice

Figure 83: Evidence of Creating Directory Test for alice

```
alice@23aF59:/sharedFolder/it$ rmdir test
rmdir: failed to remove 'test': Permission denied
alice@23aF59:/sharedFolder/it$
```

Figure 84: Evidence of Removing Directory Test for alice

```
alice@23aF59:/sharedFolder/it$ touch test.txt
touch: cannot touch 'test.txt': Permission denied
alice@23aF59:/sharedFolder/it$
```

Figure 85:Evidence of Creating test.txt File for alice

```
carol@23aF59:~$ mkdir test
mkdir: cannot create directory ‘test’: File exists
carol@23aF59:~$
```

Figure 86: Evidence of Creating Directory in Main Directory for Carol

```
carol@23aF59:~$ ls -l
total 4
drwxrwxr-x 2 carol carol 4096 Sep 20 03:40 test
carol@23aF59:~$
```

Figure 86: Evidence of Listing details for Main Directory

```
I have no name!@23aF59:~$ ls -a
.   .bash_history  .bashrc      .cache   .local    test
..  .bash_logout   .bashrc_bak  .config  .profile
I have no name!@23aF59:~$
```

Figure 87: Evidence of Files in Chroot Jail Directory

```
I have no name!@23aF59:~$ ls -l
total 4
drwxrwxr-x 2 1003 1003 4096 Sep 20 07:40 test
I have no name!@23aF59:~$
```

Figure 88: Evidence of File Details in Chroot Jail Directory

Home Mount Testing Method/Evidence

The home mount testing method refers to testing the process of mounting a file system or storage device in a home directory of the user. Figure 90, shows the user is Bob, and Figures 91-93 tests his permissions and prove he can access and create files. This is clear as no errors appear. On the other hand, Alice does the same tests and receives errors like “Command not found” or “No such file or directory” as shown in Figures 96-98.

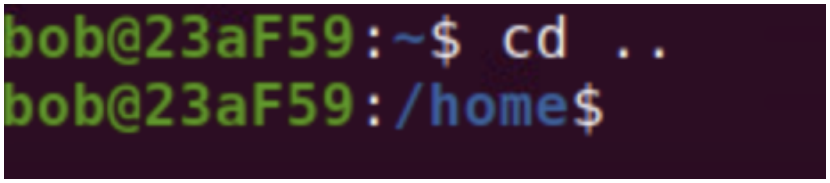

Figure 89: Evidence of Changing Directory to Upper Directory

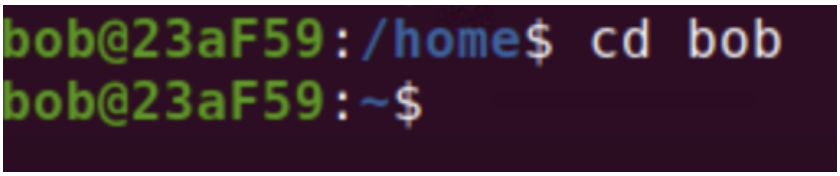

Figure 90: Evidence of Changing Directory to Home Directory

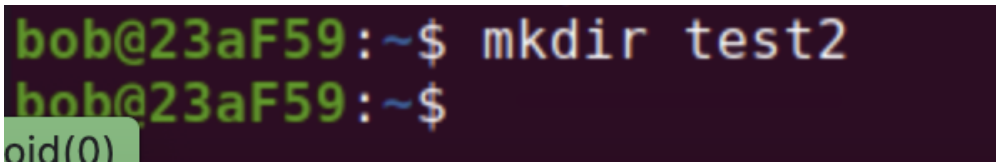

Figure 91: Evidence of Making Directory in Home Profile

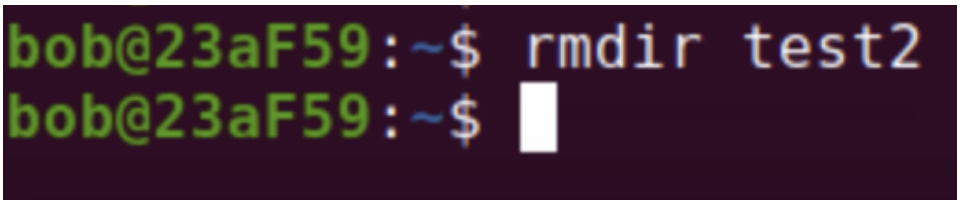

Figure 92: Evidence of Removing Directory in Home Profile

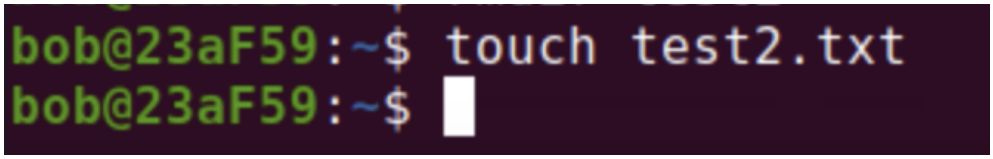

Figure 93: Evidence of Creating Text File in Home Profile

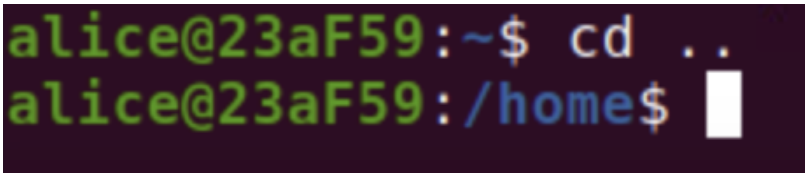

Figure 94: Evidence of Changing Directory to Upper Directory for Alice

Figure 95: Evidence of Changing Directory to Home Directory for Alice

Figure 96: Evidence of Making Directory in Alice Home Profile

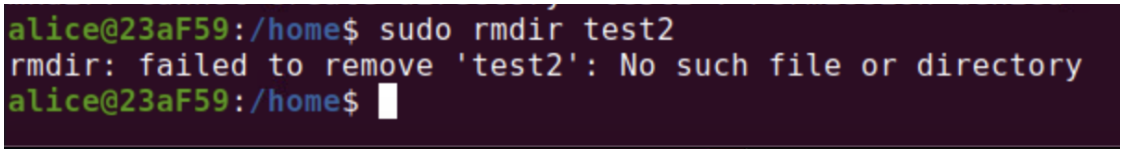

Figure 97: Evidence of Removing Directory in Alice's Home Profile

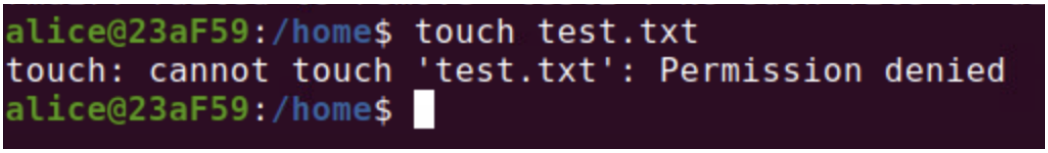

Figure 98: Evidence of Creating Text File in Home Profile

Works Cited

Avi. (2024, January 31). How to List All Groups in Linux. DevOps Blog; DevOps Blog. https://kodekloud.com/blog/how-to-list-all-groups-in-linux/

Bash mkdir and subfolders. (n.d.). Stack Overflow. https://stackoverflow.com/questions/9242163/bash-mkdir-and-subfolders

Cipriani, T. (2020, January 12). UNIX Permissions for Dummies - Tyler Cipriani. Tylercipriani.com. https://tylercipriani.com/blog/2020/01/12/unix-permissions-for-dummies/

Exploring /dev/random vs. /dev/urandom and /dev/zero vs. /dev/null. (2010, September 15). Unixbhaskar's Blog; Unixbhaskar's Blog. https://unixbhaskar.wordpress.com/2010/09/15/exploring-devrandom-vs-devurandom-and-devzero-vs-devnull/

Gite, V. (2017, August 2). *Understanding /etc/passwd File Format - nixCraft*. NixCraft. https://www.cyberciti.biz/faq/understanding-etcpasswd-file-format/

Hands, N. (2024). Lab Manual: Access Control. Spring 2024 CNIT 27000 LEC. https://purdue.brightspace.com/d2l/le/content/1094444/viewContent/17496537/View

How to Create Groups in Linux (groupadd Command). (2019, October 7). Linuxize.com. https://linuxize.com/post/how-to-create-groups-in-linux/

How to Grant Admin Privileges to a User in Linux. (2023, December 8). GeeksforGeeks. https://www.geeksforgeeks.org/how-to-grant-admin-privileges-to-a-user-in-linux/

Linux Commands - GeeksforGeeks. (2019, February 10). GeeksforGeeks. https://www.geeksforgeeks.org/linux-commands/

Manage User and Groups in Linux - TecAdmin. (2018, March 29). TecAdmin. https://tecadmin.net/tutorial/linux/manage-users-and-groups/

Mount command in Linux with Examples. (2019, March 8). GeeksforGeeks. https://www.geeksforgeeks.org/mount-command-in-linux-with-examples/

Pathak, N. (2020, March 3). *Methods to Enable or Disable Root Login in Linux - LinuxForDevices.* https://www.linuxfordevices.com/tutorials/linux/enable-disable-root-login-in-linux

Purdue University. (2016). CNIT 270 - *Cybersecurity Fundamentals. Linux Command Line Quick Reference Sheet.* Nicole Hands. https://purdue.brightspace.com/d2l/le/content/1094444/viewContent/17496524/View

Rosencrance, L. (2023, September). *What is the Principle of Least Privilege (POLP)?* SearchSecurity. https://www.techtarget.com/searchsecurity/definition/principle-of-least-privilege-POLP

sbiradar. (n.d.). How to create, delete, and modify groups in Linux. Enable Sysadmin. https://www.redhat.com/sysadmin/linux-groups

tcarriga. (2020, October 15). *Linux permissions: SUID, SGID, and sticky bit*. Enable Sysadmin. https://www.redhat.com/sysadmin/suid-sgid-sticky-bit

Wu, X. (2019, November 17). Create New User with Temporary Password on Linux. Xiaoliang's Blog. https://xwu64.github.io/2019/11/16/Create-New-User-with-Temporary-Password-on-Linux/

Zhang, E. (2023, May 5). *What Is Role-Based Access Control (RBAC)? Examples, Benefits, and More*. Digital Guardian. https://www.digitalguardian.com/blog/what-role-based-access-control-rbac-examples-benefits-and-mo