REPORT

Varunkumar Pande - 1001722538

Description of how code works:

Algorithm:

1. Validate the number of inputs provided to the program.
2. Based on the command entered validate the data against regex, if data is valid execute the command, else throw an error.
3. Use SQL parameterization to avoid SQL injection while persisting the data in sqlite3 database.

Linux does not allow execution of set-UID scripts directly hence, I had to create a C-program which is made a SET-UID root program. The user uses this C program to interact with the python script, where the core logic is implemented.

By using try-except blocks I have managed to successfully de-escalate the privileges at the end of program. As soon as the program begins I reduce the privilege to that of the user running the program, so as to avoid any threats due to privileged execution:

```
if __name__ == '__main__':
    os.seteuid(os.getuid())
    # check for number of arguments
    if len(sys.argv) == 1:
        print('Command Help!')
```

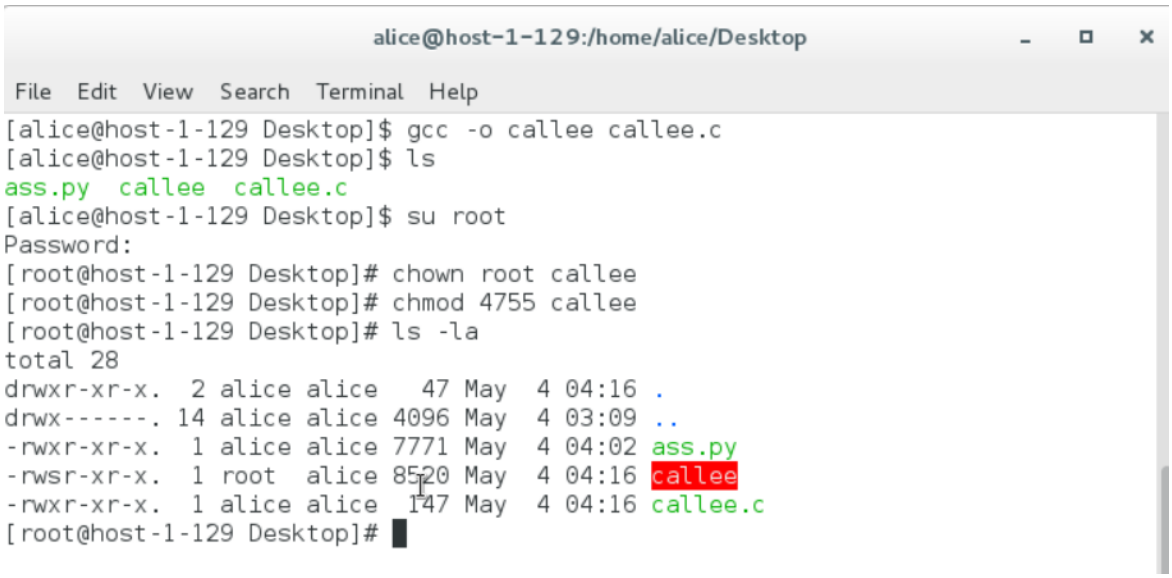
Similar approach is used when I escalate the privilege to perform a privileged function like writing to database or logging and then de-escalate it post the function is done executing.

```
def logger(log_data):
    try:
        os.seteuid(0)
        file_handler = open(str(os.getcwd())+'/logger.txt','a')
        file_handler.write(str(datetime.datetime.now()) + " " + str(os.getuid()) + log_data + "\n")
    except Exception as msg:
        error_func(msg)
    finally:
        os.seteuid(os.getuid())
        file_handler.close()
```

Steps to execute the program:

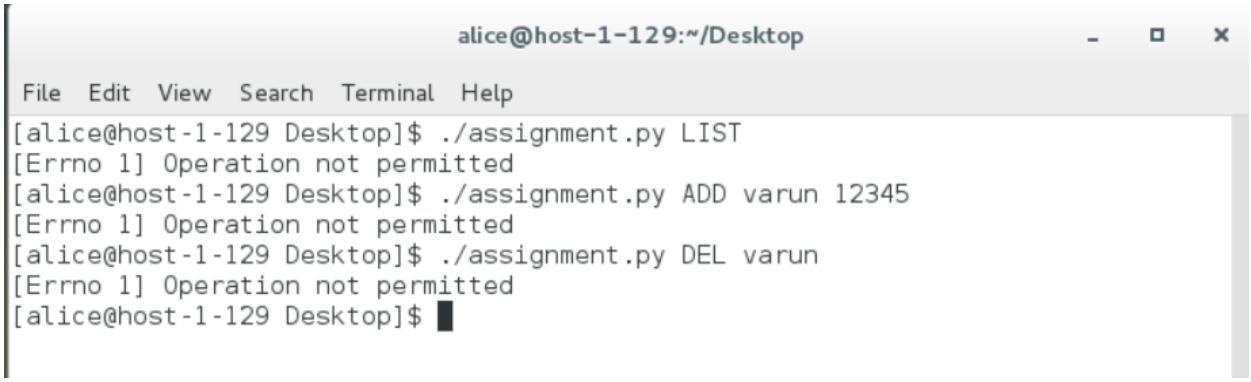
1. Since Linux has security feature that does not allow us to execute set-UID based scripts. We need to create a C program wrapper to call the python script, which basically needs to be made a set-UID root program and the parameters passed to it are simply forwarded to python program. The code for which is shown below:

```
// used as a wrapper to invoke a set-UID root python process

#include <unistd.h>

void main(int argc, char **argv){

   execve("./assignment.py", argv, NULL);

}
```



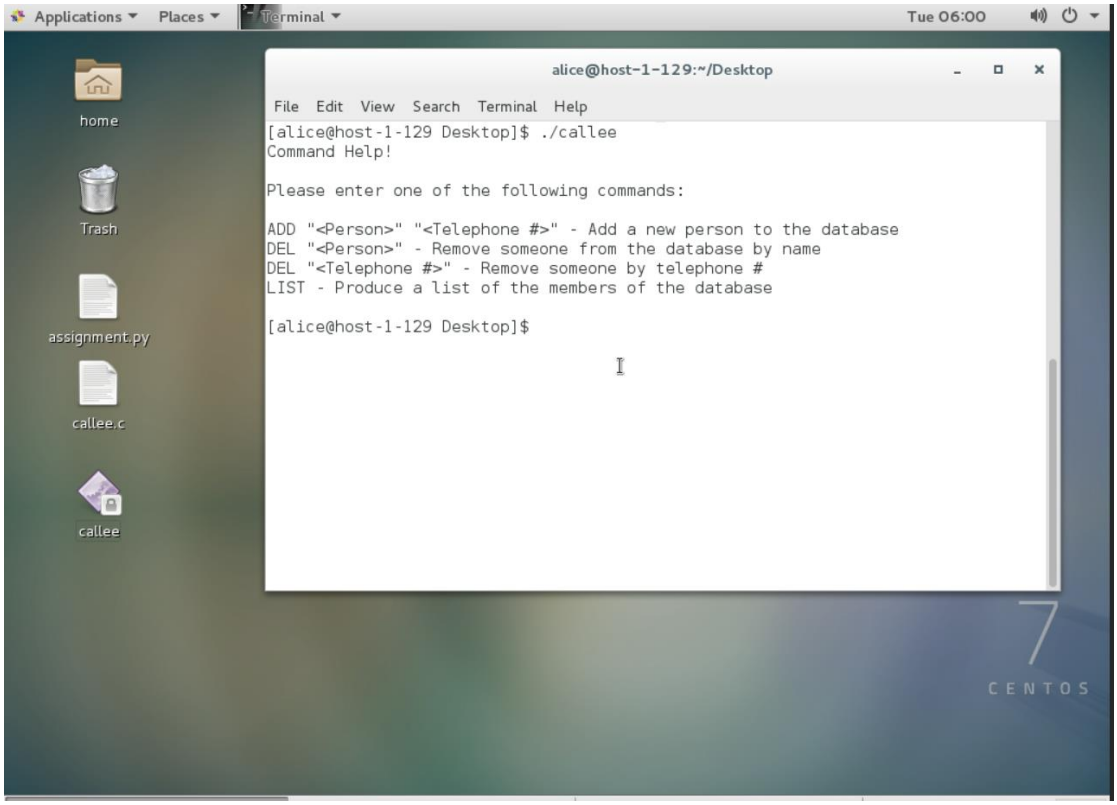
In the below screenshot we can see that the callee program is SET-UID to root, so when the program is invoked, it runs with root privileges which in turn invokes the python program with root privileges.

2. The above step is a required step in order for smooth functioning of the directory program. Else we will get permission issues while executing the directory program, as shown below:

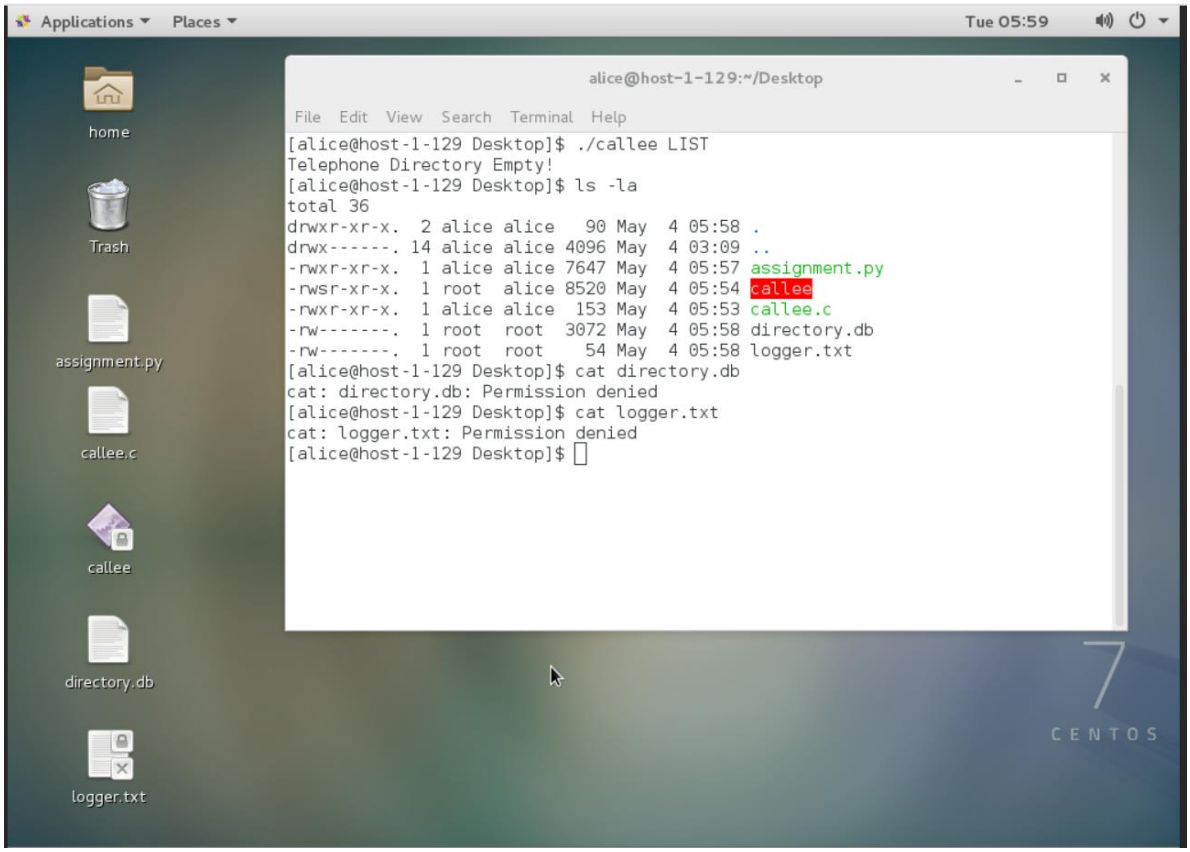


Running commands:

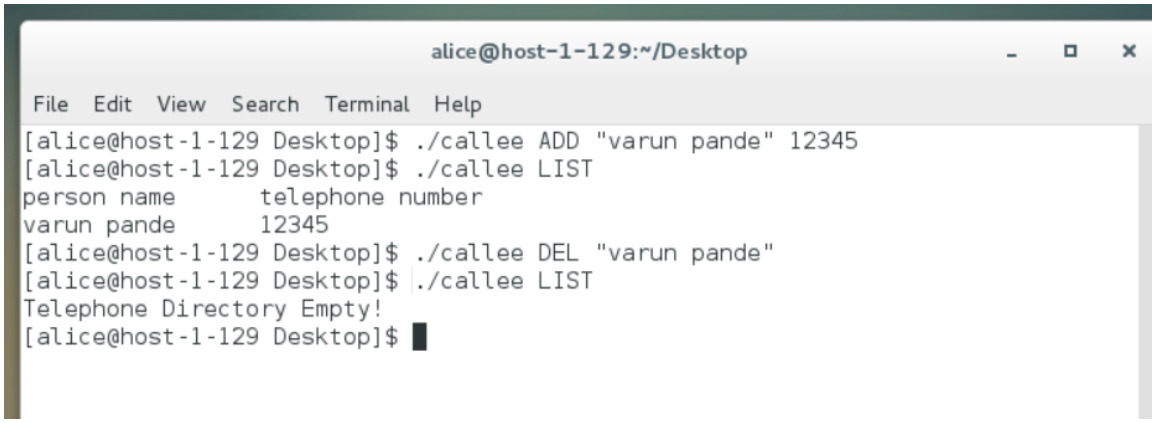
1. Running the directory program without arguments lead to the help screen:



2. Any execution of valid commands will lead to creation of the “directory.db” and “logger.txt” files which are used to persist the data and log commands respectively. In the screenshot below we can also see that both the files are access controlled and are in-accessible to to the current user.

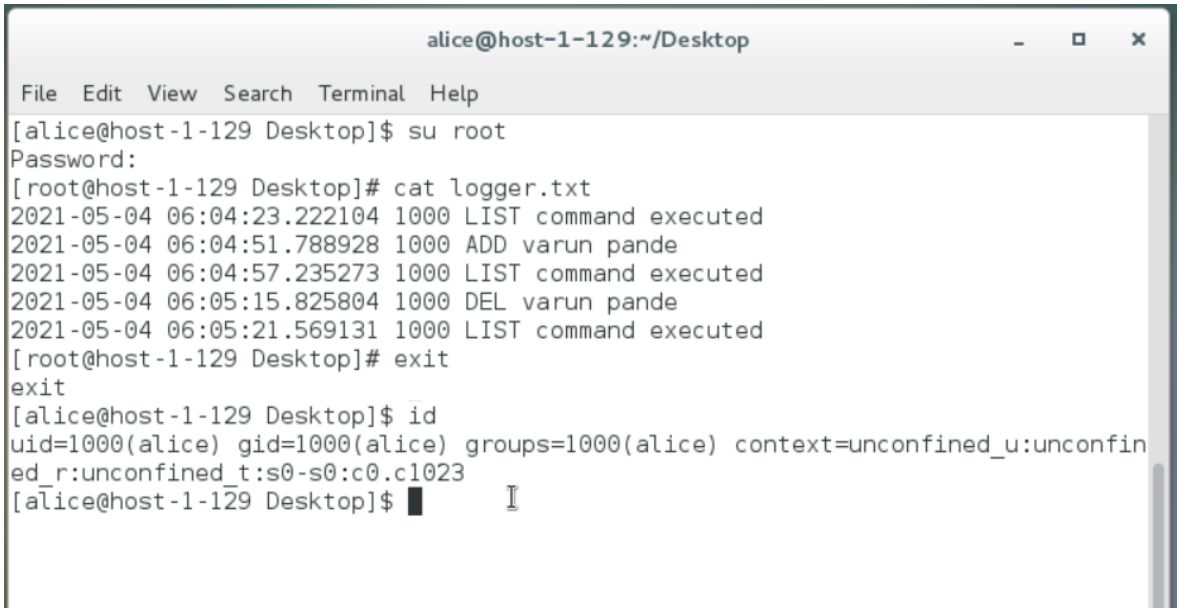


3. Running other commands like ADD user and DEL user also works as expected:



Check logs:

Since the log files are only root accessible, we need to login as root to check the logs. The log file stores information of each executed command with timestamp and real user-id of the person executing the program, as shown below:



In the above screenshot we can see that the id of “Alice” is 1000 which gets logged in the logger.

Assumptions I have made:

The user should be able to make the c-program Set-UID to root. If the user downloads the compiled c-program it should have executing permissions enabled and should be SET-UID to root.

Pros/Cons of my approach:

Pros:

1. The db-file and log files are access controlled.
2. I performed privilege escalation and de-escalation based on the flow of program.
3. Made use of regex to validate the input data. Implemented proper check on the number of incoming parameters. Implemented SQL query parameterization to avoid SQL injection.

```
TelephoneListing.DB_CONNECT.execute('INSERT INTO TELEPHONE_DIRECTORY VALUES (?, ?);', (person, telephone))
TelephoneListing.DB_CONNECT.commit()
```

```
db_cursor.execute('SELECT NAME FROM TELEPHONE_DIRECTORY WHERE PHONE=(?);', (telephone,))
```

```
person:
 TelephoneListing.DB_CONNECT.execute('DELETE FROM TELEPHONE_DIRECTORY WHERE PHONE=(?);', (telephone,))
```

Cons:

1. While implementing the C-wrapper I could have used a md5 checksum to validate that only my “assignment.py” file gets executed when the “callee” program is run.
2. Use of other database system that provide more granular access to data and allow user logins, encryption of data.