MY_SUDO BOOTSTRAP

MAKE ME A SANDWICH

MY_SUDO BOOTSTRAP

1. Unix accounting 101

Don't overlook this part. Being able to effortlessly create and switch user account will be helpful to test your program during the project.

Unix Users and Group creation

Let's create 3 users (each with their own password): *toto*, *tata*, *tutu* and 2 groups: *pedago* and *student*
And assign groups to those users in such a way that:
- *toto* is part of the *student* group
- *tata* is part of the *pedago* group
- *tutu* is part of the *student* and *pedago* group

- ✓ You may need to take a look to the manual of the following commands: `useradd`, `groupadd` , `usermod`, `passwd`
- ✓ A privileged account (root) may be required to use those commands properly.
- ✓ Do you know which files on your machine store information about users, groups and user's passwords ?

Substitute User

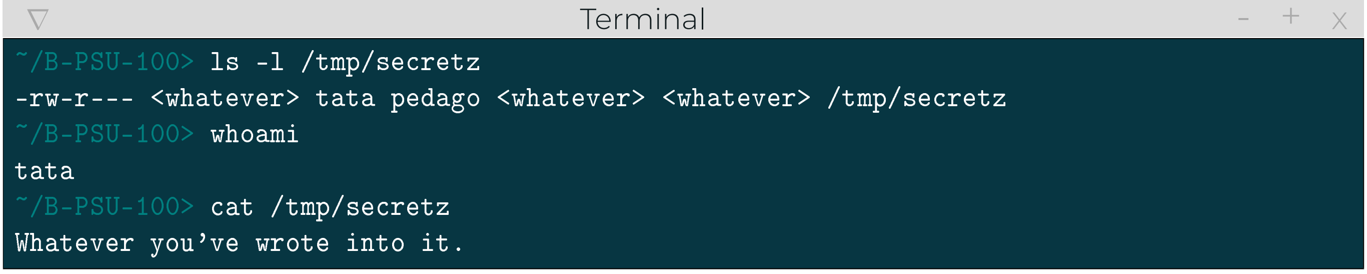
As you may know processes are owned by users and groups on Unix system in the same way files are.
By default processes are owned by the user who ran them.

Now you can open multiple terminals and switch to one of those previously created users by using the `su` command.

By running the `ps -aux` command as any user, you should notice that you now have shell processes owned by those users running on your computer.
You can also run the `id` or `whoami` command inside those shells to see who is owning the process.

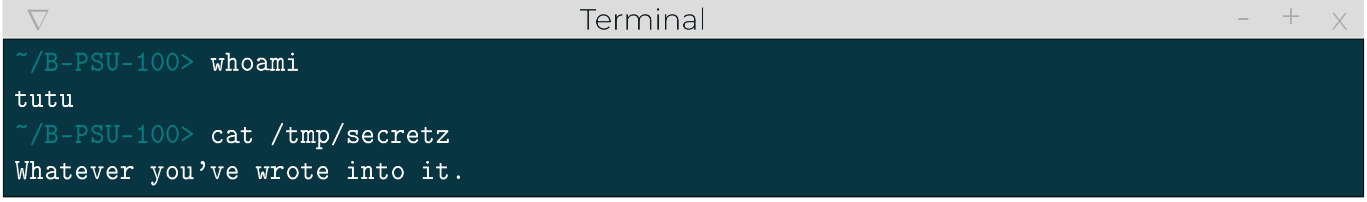
Permissions

Switch to the user *tata* and create a file `/tmp/secretz`.
Set the permissions on this file so that you have something similar to this:



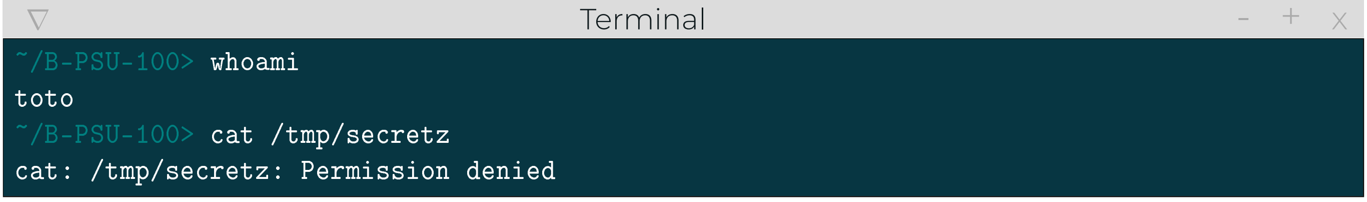
```
~/B-PSU-100> ls -l /tmp/secretz
-rw-r---- <whatever> tata pedago <whatever> <whatever> /tmp/secretz
~/B-PSU-100> whoami
tata
~/B-PSU-100> cat /tmp/secretz
Whatever you've wrote into it.
```

Now switch to the user *tutu*. This user can also read the content of this file.



```
~/B-PSU-100> whoami
tutu
~/B-PSU-100> cat /tmp/secretz
Whatever you've wrote into it.
```

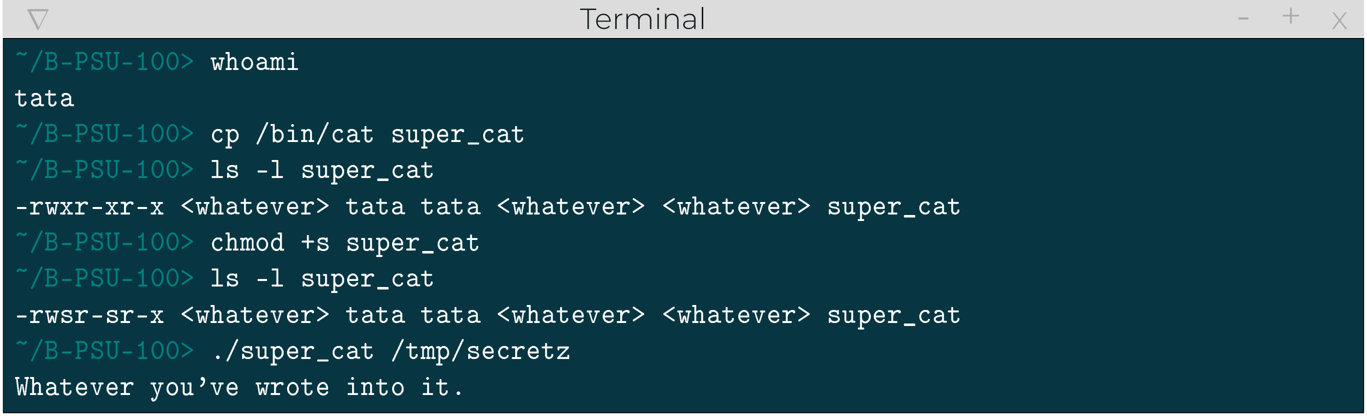
Now switch to the user *toto*. This user cannot read the content of this file. Can you say why ?



```
~/B-PSU-100> whoami
toto
~/B-PSU-100> cat /tmp/secretz
cat: /tmp/secretz: Permission denied
```

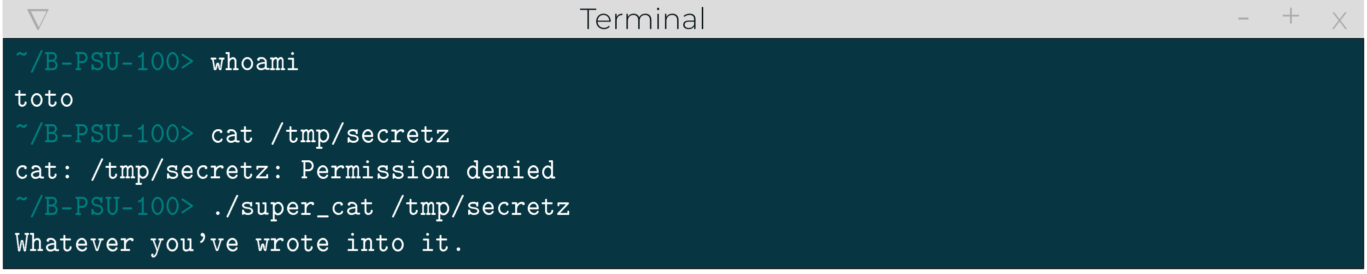
SUID/SGID rights

Switch to the user tata. And make a copy of the `cat` program named super_cat into the current directory.
Then set the suid/sgid rights on supercat with the `chmod` command.



```
~/B-PSU-100> whoami
tata
~/B-PSU-100> cp /bin/cat super_cat
~/B-PSU-100> ls -l super_cat
-rwxr-xr-x <whatever> tata tata <whatever> <whatever> super_cat
~/B-PSU-100> chmod +s super_cat
~/B-PSU-100> ls -l super_cat
-rwsr-sr-x <whatever> tata tata <whatever> <whatever> super_cat
~/B-PSU-100> ./super_cat /tmp/secretz
Whatever you've wrote into it.
```

Now switch back to the user *toto*.



```
~/B-PSU-100> whoami
toto
~/B-PSU-100> cat /tmp/secretz
cat: /tmp/secretz: Permission denied
~/B-PSU-100> ./super_cat /tmp/secretz
Whatever you've wrote into it.
```

✓ So what was the effect of setting the “suid/sgid” rights on super_cat with `chmod +s` ?
✓ What are the implications security wise ?

2. Execution

My Exec

Write a function named my_exec that takes 2 arguments.
The first argument is the name or path of a program to run, the second argument will be passed as the first argument of the program to run.

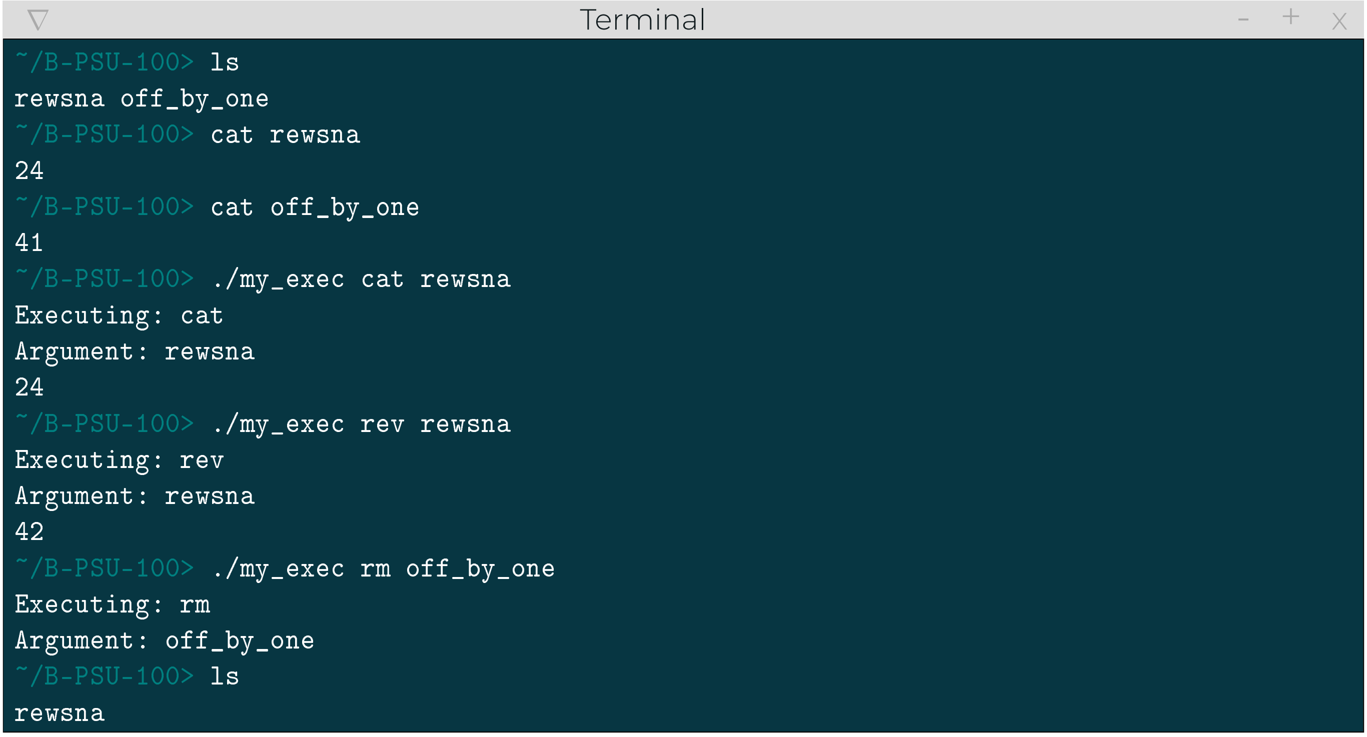
Your function will be prototyped as follow:

```c
void my_exec(char *arg0, char *arg1);
```

Your function will be tested using this main function:

```c
int main(int argc, char *argv)
{
    if (argc < 3)
        return 84;
    my_exec(argv[1], argv[2]);
}
```

For example:

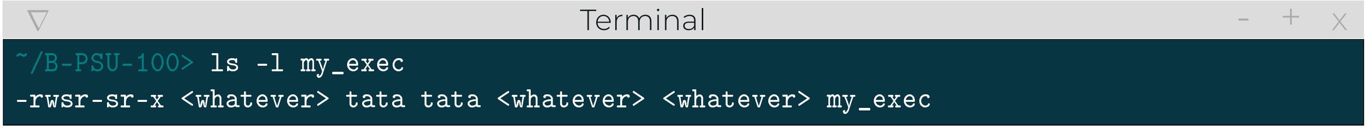


```
~/B-PSU-100> ls
rewsna off_by_one
~/B-PSU-100> cat rewsna
24
~/B-PSU-100> cat off_by_one
41
~/B-PSU-100> ./my_exec cat rewsna
Executing: cat
Argument: rewsna
24
~/B-PSU-100> ./my_exec rev rewsna
Executing: rev
Argument: rewsna
42
~/B-PSU-100> ./my_exec rm off_by_one
Executing: rm
Argument: off_by_one
~/B-PSU-100> ls
rewsna
```

```
man execlp
```

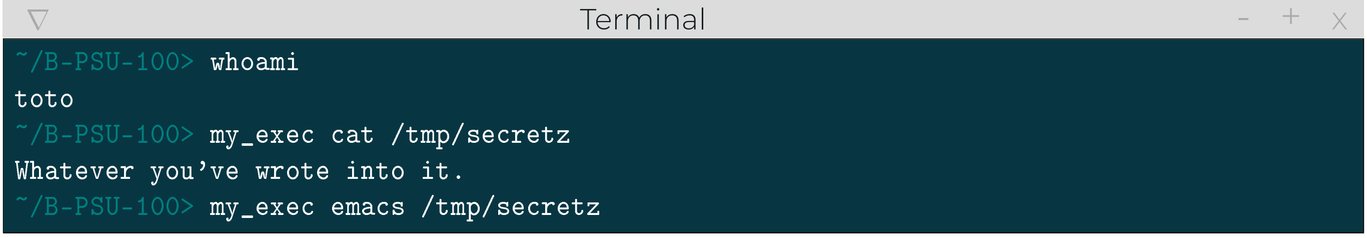
Adding the suid to our program

Set tata as the owner to your program my_exec and add the suid to its permissions.



```
~/B-PSU-100> ls -l my_exec
-rwsr-sr-x <whatever> tata tata <whatever> <whatever> my_exec
```

Now as toto



```
~/B-PSU-100> whoami
toto
~/B-PSU-100> my_exec cat /tmp/secretz
Whatever you've wrote into it.
~/B-PSU-100> my_exec emacs /tmp/secretz
```

So you now have a program on your computer that can run any program as tata.
That is not safe at all.
Think about all the good (or bad) things that can happen if this program was owned by root and had the suid right.
In the next step you will add a bit of security to it.

3. Requiring a password to run our program

Modify your previous code by adding a function check_password that:
- Ask the user for a password on the standard input
- If the user entered the correct password (the hash of the correct password to compare with the user input is passed as an argument to check_password): write "Access granted" and return 1
- Or else: write "Access denied" and return 0

Your function will be prototyped as follow:

```
int check_password(const char *password_hash)
```

Your functions will be tested with the following main function:

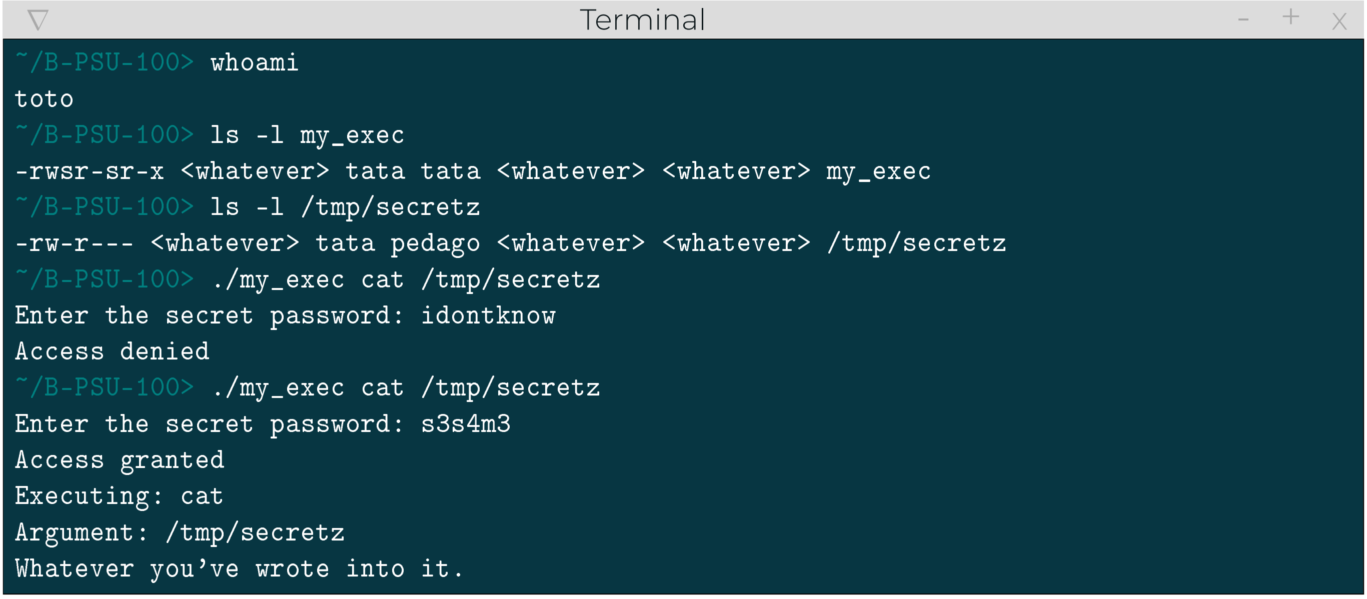
```
#define HASH "$y$j9T$0JSXQIDBQiUYZ4Y1niCGS/$qXVlMuA7Ez4hVbzNoq3FCUaJBT70qXu4330giD7ykI0"

int main (int argc, char *argv)
```

```
{
    if (argc < 3)
        return 84;
    if (check_password(HASH) == 1)
        my_exec(argv[1], argv[2]);
    else
        return 84;
}
```

✓ Allowed functions: `getline`, `crypt`, `execlp`
✓ link your program with -lcrypt

Expected output:



```
~/B-PSU-100> whoami
toto
~/B-PSU-100> ls -l my_exec
-rwsr-sr-x <whatever> tata tata <whatever> <whatever> my_exec
~/B-PSU-100> ls -l /tmp/secretz
-rw-r---- <whatever> tata pedago <whatever> <whatever> /tmp/secretz
~/B-PSU-100> ./my_exec cat /tmp/secretz
Enter the secret password: idontknow
Access denied
~/B-PSU-100> ./my_exec cat /tmp/secretz
Enter the secret password: s3s4m3
Access granted
Executing: cat
Argument: /tmp/secretz
Whatever you've wrote into it.
```

The my_exec program is a little bit safer now. But the correct password is displayed on the terminal while you type it on your keyboard.
There is for sure a way to temporarily hide your input on the terminal. You might figure it out while doing your my_sudo project.

V1

{EPITECH}