KØBENHAVNS UNIVERSITET

ITS - Assignment 2

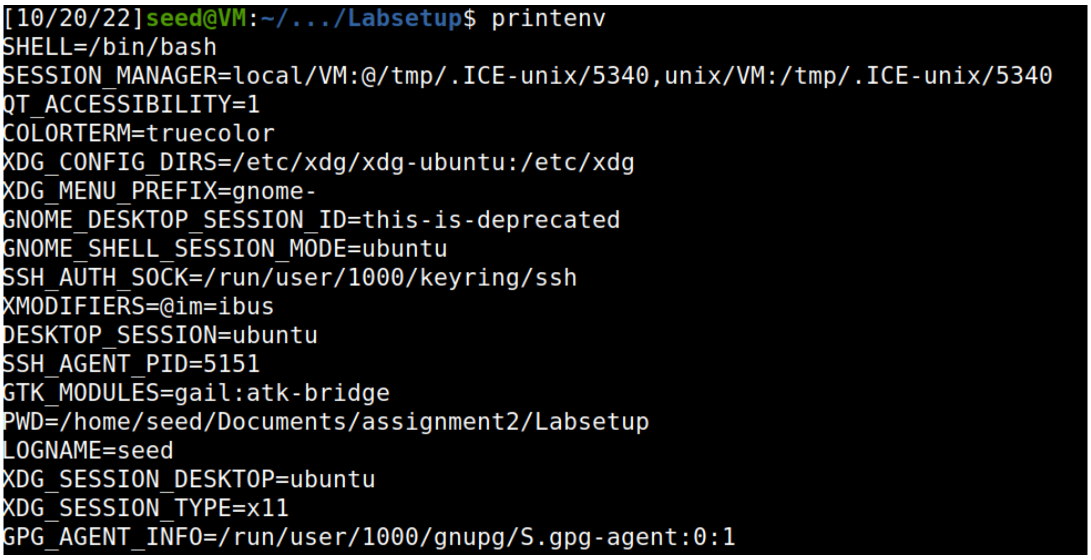
Anders Friis Persson, Oliver Meulengracht og Mikkel Willén

1. november 2022

Indhold

Task 1

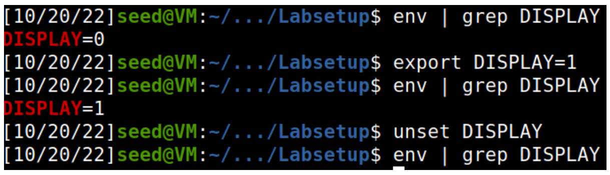
Using the "printenv"or "env"command in the terminal, we get all the eviroment variables, as seen below.



```
[10/20/22]seed@VM:~/.../Labsetup$ printenv
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/5340,unix/VM:/tmp/.ICE-unix/5340
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=5151
GTK_MODULES=gail:atk-bridge
PWD=/home/seed/Documents/assignment2/Labsetup
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
```

Figur 1: Entering the printenv program

If we write "env | grep DISPLAY", we get the values of the enviromental variable "DISPLAY". We can change or set the value with export, or remove it with unset.



```
[10/20/22]seed@VM:~/.../Labsetup$ env | grep DISPLAY
DISPLAY=0
[10/20/22]seed@VM:~/.../Labsetup$ export DISPLAY=1
[10/20/22]seed@VM:~/.../Labsetup$ env | grep DISPLAY
DISPLAY=1
[10/20/22]seed@VM:~/.../Labsetup$ unset DISPLAY
[10/20/22]seed@VM:~/.../Labsetup$ env | grep DISPLAY
```

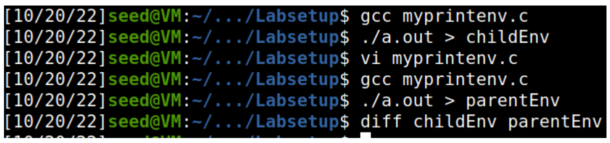
Figur 2: Using export and unset

Task 2

We got a program where we were tasked to change a line in the code seen below.

```
case 0:  /*child process*/
    printenv();
    exit(0);
default:  /*parent process*/
    //printenv();
```

We compile the code into binary using 'gcc' command, which is the compiler toolchain used to compile 'c' code. The command 'diff' compares 2 files line by line along with the changes that are needed to carry out to make them both identical. We ran and compiled the code within 'myprintenv.c' where it executed a function called 'printenv()' which prints all the enviromental variables for the child process. We then removed the function call in the child process and then wrote it in the parent process.
We compare the output of the 2 binary files using the 'diff' command. As seen by the picture below, there was no difference between the to files, and we can thereby conclude, that the child process has the same enviroment variables as the parent process.



```
[10/20/22]seed@VM:~/.../Labsetup$ gcc myprintenv.c
[10/20/22]seed@VM:~/.../Labsetup$ ./a.out > childEnv
[10/20/22]seed@VM:~/.../Labsetup$ vi myprintenv.c
[10/20/22]seed@VM:~/.../Labsetup$ gcc myprintenv.c
[10/20/22]seed@VM:~/.../Labsetup$ ./a.out > parentEnv
[10/20/22]seed@VM:~/.../Labsetup$ diff childEnv parentEnv
```

Figur 3: Comparing the difference of output of child and parent

Task 3

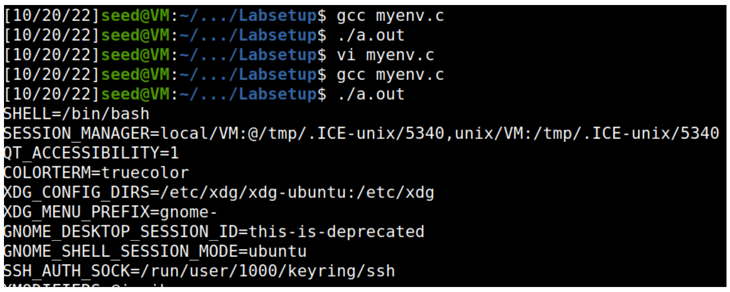
We were tasked with figuring out, how execve gets its envirometal variables. Below we have to versions of a call to execve.

```
execve("/usr/bin/env", argv, NULL);
```

vs

```
execve("/usr/bin/env", argv, environ);
```

We were given a program called myenv.c a ran the program with these two lines. From this we



```
[10/20/22]seed@VM:~/.../Labsetup$ gcc myenv.c
[10/20/22]seed@VM:~/.../Labsetup$ ./a.out
[10/20/22]seed@VM:~/.../Labsetup$ vi myenv.c
[10/20/22]seed@VM:~/.../Labsetup$ gcc myenv.c
[10/20/22]seed@VM:~/.../Labsetup$ ./a.out
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/5340,unix/VM:/tmp/.ICE-unix/5340
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
```

Figur 4: Commands for tests of execve

could conlude, that execve gets it enviromental variables from the 3rd argument in the function call. If we check the manpage for execve, we can confirm this. Below is the definition for execve.

```
int execve(const char *pathname, char *const argv[], char *const envp[]);
```

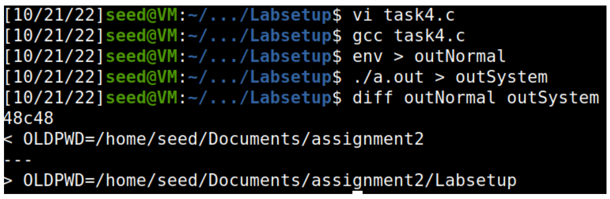
Task 4

We were tasked with verifying that the new program gets its enviroment variables from the calling process, when using system(). We wrote the following program and ran it.

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    system("/usr/bin/env");
    return 0;
}
```

We then saved the output in a file and saved the output of "env"to another file, then called diff, to see the differences.

All the enviroment variables are the same except for OLDPWD. We assume this variable is changed after the program is created, and that all envoriment varibales are copied from the calling process.



```
[10/21/22]seed@VM:~/.../Labsetup$ vi task4.c
[10/21/22]seed@VM:~/.../Labsetup$ gcc task4.c
[10/21/22]seed@VM:~/.../Labsetup$ env > outNormal
[10/21/22]seed@VM:~/.../Labsetup$ ./a.out > outSystem
[10/21/22]seed@VM:~/.../Labsetup$ diff outNormal outSystem
48c48
< OLDPWD=/home/seed/Documents/assignment2
---
> OLDPWD=/home/seed/Documents/assignment2/Labsetup
```

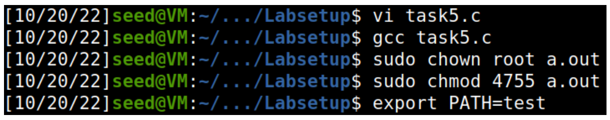
Figur 5: Running the program, and getting the variables

Task 5

We were tasked with figuring out, if Set-UID programs process inherits the enviroment varibales from the users process. We wrote the following program.

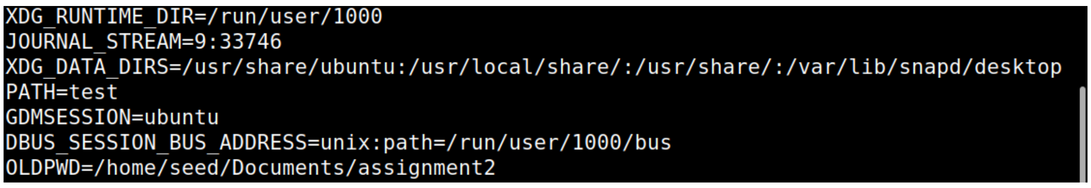
```
#include <stdio.h>
#include <stdlib.h>

extern char** environ;

int main(){
    int i = 0;
    while (environ[i] != NULL) {
        printf("%s\n", environ[i]);i++;
    }
}
```

We compiled the program, gave it root privileges, and changed one of the enviroment variables. We got the following output from our program: We can see the varible PATH has been changed



```
[10/20/22]seed@VM:~/.../Labsetup$ vi task5.c
[10/20/22]seed@VM:~/.../Labsetup$ gcc task5.c
[10/20/22]seed@VM:~/.../Labsetup$ sudo chown root a.out
[10/20/22]seed@VM:~/.../Labsetup$ sudo chmod 4755 a.out
[10/20/22]seed@VM:~/.../Labsetup$ export PATH=test
```

Figur 6: Compiling and given the program root privileges



```
XDG_RUNTIME_DIR=/run/user/1000
JOURNAL_STREAM=9:33746
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share/:/usr/share/:/var/lib/snapd/desktop
PATH=test
GDMSESSION=ubuntu
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
OLDPWD=/home/seed/Documents/assignment2
```

Figur 7: Running the program, and getting the environmental variables

and is inherited.

Task 6

We were tasked with making the terminal run another program, than originally intented. We wrote the following program.

```
int main(){
    system("ls");
    return 0;
}
```

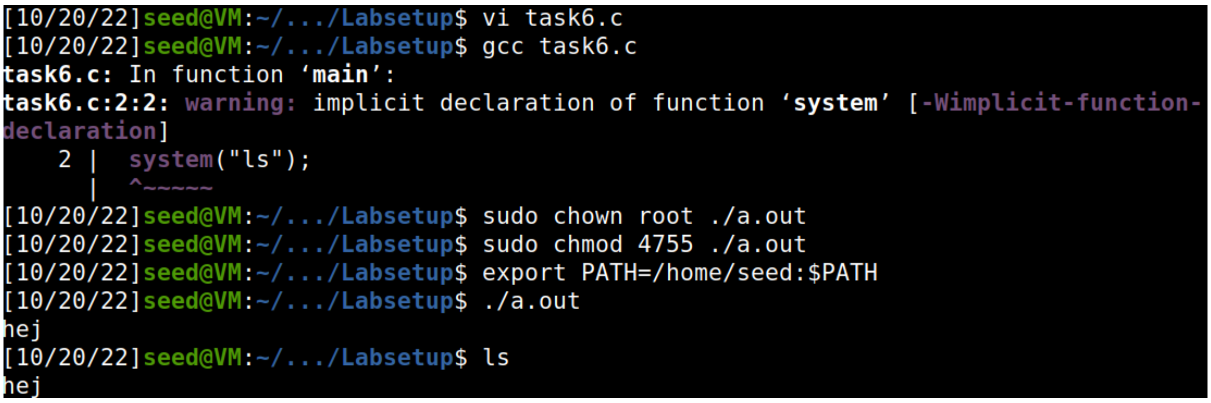
We then compiled that program and gave it root privileges. We made a program bash program called ls in the home folder, which looked like this:

```
echo "hej"
```

and changed the PATH variable.

```
PATH=/home/seed:$PATH
```

Running our first program, we get "hej"as output, as seen on the picture below.



```
[10/20/22]seed@VM:~/.../Labsetup$ vi task6.c
[10/20/22]seed@VM:~/.../Labsetup$ gcc task6.c
task6.c: In function ‘main’:
task6.c:2:2: warning: implicit declaration of function ‘system’ [-Wimplicit-function-
declaration]
    2 |  system("ls");
      |  ^~~~~~
[10/20/22]seed@VM:~/.../Labsetup$ sudo chown root ./a.out
[10/20/22]seed@VM:~/.../Labsetup$ sudo chmod 4755 ./a.out
[10/20/22]seed@VM:~/.../Labsetup$ export PATH=/home/seed:$PATH
[10/20/22]seed@VM:~/.../Labsetup$ ./a.out
hej
[10/20/22]seed@VM:~/.../Labsetup$ ls
hej
```

Figur 8: Changing the PATH and runnign the program and LS

Even when running linux version of ls, we get the output "hej".

Short Questions - 1

The categories are:

1. **The "What you know":** Which includes things you have remembered mentally, such as passwords, PINs and passphrases.
2. **The "What you have":** 'Uses a computer or hardware token physically possessed (ideally, difficult to replicate), often holding a cryptographic secret; or a device having hard-to-mimic physical properties.'
3. **The "What you are":** Includes physical biometrics, such as fingerprints: where related methods involve behavioral biometrics or distinguishing behavioral patterns.

Short Questions - 2

Passwords are securely stored in a file by storing them in pairs of user id and hashed password using a one-way hash function. To protect against dictionary attacks, it is common practice to salt passwords before they are hashed. Instead of storing $h_i = H(p_i)$ upon registration of each password p_i, the system selects a random t-bit value s_i as salt and stores:

$$(u_i, s_i, H(p_i, s_i))$$

before hashing with p_i, s_i concatenated. The userid is u_i in this case. This ensures that the salt alters the password in a deterministic manner before hashing. Salting has the added benefit of resulting in different password hashes in the system hash file for two users who use the same password.

Short Questions - 3

The main reason for using RBAC is that it's easy in larger scales. Ideally you don't have too many roles, and it's just a question of granting every user the correct role. Using DAC, you would have to specify all the resources the new users need access to and add them. MAC is mostly used in millitary systems.

Short Questions - 4

Using cross-site scripting(XSS), the attacker is able to steal session cookies and then pretend to be the user. XSS injects malicious HTML tags or scripts into web pages, causing HTML rendering on the user agent's browser to perform actions that neither legitimate sites nor users intended.