INFORMATION SECURITY LAB 01

Name: Pavan R Kashyap SRN: PES1UG20CS280
6th Semester E section

TASK 1: Manipulating environment variables

In this task, we understand what environment variables are and how they are set (and unset).

The printenv command is used to print the list of the system's environment variables. Details like the present working directory, user, shell script etc. are displayed to the user.

```
PES1UG20CS280:Pavan -$printenv
SHELL=/bin/bash
PWD=/home/seed/Labsetup
LOGNAME=seed
MOTD_SHOWN=pam
HOME=/home/seed
SSH_CONNECTION=10.0.0.2 6927 10.0.0.149 22
TERM=xterm-256color
USER=seed
SHLVL=1
PS1=PES1UG20CS280:Pavan -$
SSH_CLIENT=10.0.0.2 6927 22
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
SSH_TTY=/dev/pts/1
_=/usr/bin/printenv
OLDPWD=/home/seed
```

The present working directory (PWD) or the current working directory can be obtained by using the command specified below-

```
PES1UG20CS280:Pavan -$printenv PWD
/home/seed/Labsetup
PES1UG20CS280:Pavan -$
```

The command indicates that the user is in the /home/seed/Labsetup directory currently.

In this subtask, we define our own environment variable and set it. We use the export command to set the env. Variable 'foo' (user defined and set to PES1UG20CS280). Once set, we print that environment variable and we see the corresponding value it holds.

Once unset, the environment variable is removed and therefore, the second printenv command returns nothing.

```
PES1UG20CS280:Pavan -$
PES1UG20CS280:Pavan -$export foo='PES1UG20CS280'
PES1UG20CS280:Pavan -$printenv foo
PES1UG20CS280
PES1UG20CS280:Pavan -$unset foo
PES1UG20CS280:Pavan -$printenv foo
PES1UG20CS280:Pavan -$
```

INFORMATION SECURITY LAB 01

Name: Pavan R Kashyap SRN: PES1UG20CS280
6th Semester E section

Task 2 : Passing Environment Variables from parent process to child process

In this task, we see if the child process inherits the environment variables of the parent process.

In myprintenv.c, the parent process is forked to generate a child process and the printenv function is used to print the environment variables of the child process. The output is redirected to a file 'child' and its contents are observed below-

```
PES1UG20CS280:Pavan -$gcc myprintenv.c
PES1UG20CS280:Pavan -$./a.out > child
```

```
PES1UG20CS280:Pavan -$cat child
SHELL=/bin/bash
PWD=/home/seed/Labsetup
LOGNAME=seed
MOTD_SHOWN=pam
HOME=/home/seed
SSH_CONNECTION=10.0.0.2 6927 10.0.0.149 22
TERM=xterm-256color
USER=seed
SHLVL=1
PS1=PES1UG20CS280:Pavan -$
SSH_CLIENT=10.0.0.2 6927 22
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
SSH_TTY=/dev/pts/1
OLDPWD=/home/seed
_=./a.out
```

When the printenv() line is commented (in the child process section of the source code) , the control returns back to the parent process (after exit(0) in child process is called). The environment variables of the parent process are then printed and the result is redirected to the 'parent' file.

```
PES1UG20CS280:Pavan -$a.out > parent
```

```
PES1UG20CS280:Pavan -$cat parent
SHELL=/bin/bash
PWD=/home/seed/Labsetup
LOGNAME=seed
MOTD_SHOWN=pam
HOME=/home/seed
SSH_CONNECTION=10.0.0.2 6927 10.0.0.149 22
TERM=xterm-256color
USER=seed
SHLVL=1
PS1=PES1UG20CS280:Pavan -$
SSH_CLIENT=10.0.0.2 6927 22
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
SSH_TTY=/dev/pts/1
OLDPWD=/home/seed
_=./a.out
```

In order to identify if the parent and child processes have different ENV variables, we find the difference between the two file contents. We observe that both the files are the same (there is no difference). This indicates to us that the child process does indeed inherit the ENV variables of the parent process.

```
PES1UG20CS280:Pavan -$diff child parent
PES1UG20CS280:Pavan -$
```

INFORMATION SECURITY LAB 01

Name: Pavan R Kashyap SRN: PES1UG20CS280
6th Semester E section

Task 3 Environment variables and execve()

Execve() command overwrites the contents of the old program(process) with what is currently being executed. If environment variables of the calling process have to be retained, it must be explicitly mentioned in the execve command. It does not follow the inheritance scheme as done in task2.

The third argument of the code is changed from NULL to environ (global variable) and both are executed to see the difference.

```
  GNU nano 4.8
#include <unistd.h>

extern char **environ;

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

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

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

  return 0 ;
}
```

We see that when the third argument is NULL, there are no environment variables displayed (the calling process' ENV variables are not retained).

Similarly, when the third argument is set to environ, all environment variables are displayed (the calling process' ENV variables are retained and displayed).

```
PES1UG20CS280:Pavan -$gcc myenv.c
PES1UG20CS280:Pavan -$./a.out
PES1UG20CS280:Pavan -$nano myenv.c
PES1UG20CS280:Pavan -$PES1UG20CS280:Pavan -$
PES1UG20CS280:Pavan -$gcc myenv.c
PES1UG20CS280:Pavan -$./a.out
SHELL=/bin/bash
PWD=/home/seed/Labsetup
LOGNAME=seed
MOTD_SHOWN=pam
HOME=/home/seed
SSH_CONNECTION=10.0.0.2 7019 10.0.0.149 22
TERM=xterm-256color
USER=seed
SHLVL=1
PS1=PES1UG20CS280:Pavan -$
SSH_CLIENT=10.0.0.2 7019 22
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
SSH_TTY=/dev/pts/2
OLDPWD=/home/seed
_=./a.out
PES1UG20CS280:Pavan -$
```

INFORMATION SECURITY LAB 01

Name: Pavan R Kashyap SRN: PES1UG20CS280
6th Semester E section

TASK 4 Environment Variables and system()

In this task, we use the system() command to display the ENV variables. The ENV variables of the calling process are passed on to the new program; therefore, we are able to display all the ENV variables of the calling process when sysenv.c is executed.

```
PES1UG20CS280:Pavan -$nano sysenv.c
PES1UG20CS280:Pavan -$gcc sysenv.c -o sysenv
PES1UG20CS280:Pavan -$./sysenv
USER=seed
SSH_CLIENT=10.0.0.2 7019 22
SHLVL=1
MOTD_SHOWN=pam
HOME=/home/seed
OLDPWD=/home/seed
SSH_TTY=/dev/pts/2
PS1=PES1UG20CS280:Pavan -$
LOGNAME=seed
_=./sysenv
TERM=xterm-256color
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
SHELL=/bin/bash
PWD=/home/seed/Labsetup
SSH_CONNECTION=10.0.0.2 7019 10.0.0.149 22
PES1UG20CS280:Pavan -$
```

INFORMATION SECURITY LAB 01

Name: Pavan R Kashyap SRN: PES1UG20CS280
6th Semester E section

TASK 5 Environment variables and Set-UID programs

Set-UID programs are used to temporarily escalate the privileges of a user. Set-UID changes the effective UID (of the user) to that of the RUID of the resource that is being accessed. This temporarily escalates the privilege by allowing the user (who previously did not have the permission) to access that resource (RUID= EUID).

Chmod 4755 is used to escalate the seed user's Euid to root's Ruid (in this case). Once done, the seed user has read, write and execute permissions (7).

Chown is used to change the owner of a given file. Setuidenv.c is created by the seed user and its ownership (of the executable) is changed to that of root by the chown command.

```
PES1UG20CS280:Pavan -$nano setuidenv.c
PES1UG20CS280:Pavan -$gcc setuidenv.c -o setuid
PES1UG20CS280:Pavan -$sudo chown root setuid
[sudo] password for seed:
PES1UG20CS280:Pavan -$sudo chmod 4755 setuid
PES1UG20CS280:Pavan -$ls -l setuid
-rwsr-xr-x 1 root seed 16768 Jan 18 04:17 setuid
PES1UG20CS280:Pavan -$
```

The ls -l command indicates that setuid executable is owned by the root user (with super user privilege available as seen with s in the command above).

Environment variables are set using the export command. The Set-UID's ENV. variables are inherited from the user's process (including the new ENV. Variables that have been set by the user's process).

The resulting ENV. Variables are displayed below. Variables set by the user like task5 and PATH are also displayed as shown below-

```
PES1UG20CS280:Pavan -$export PATH=/home/seed:$PATH
PES1UG20CS280:Pavan -$export LD_LIBRARY_PATH=/home/seed:$LD_LIBRARY_PATH
PES1UG20CS280:Pavan -$export task5=PES1UG20CS280
PES1UG20CS280:Pavan -$./setuid
SHELL=/bin/bash
PWD=/home/seed/Labsetup
LOGNAME=seed
task5=PES1UG20CS280
MOTD_SHOWN=pam
HOME=/home/seed
SSH_CONNECTION=10.0.0.2 7211 10.0.0.149 22
TERM=xterm-256color
USER=seed
SHLVL=1
PS1=PES1UG20CS280:Pavan -$
SSH_CLIENT=10.0.0.2 7211 22
PATH=/home/seed:
SSH_TTY=/dev/pts/2
OLDPWD=/home/seed
_=./setuid
PES1UG20CS280:Pavan -$
```

INFORMATION SECURITY LAB 01

Name: Pavan R Kashyap SRN: PES1UG20CS280

6th Semester E section

TASK 6 THE PATH Environment variable and Set-UID programs

The system() call has the ls command which lists all the contents of the given directory. The system command looks for the PATH where the ls command file is present (/bin/sh) and executes the ls command . The PATH resolution is present, and therefore the corresponding ls command is executed and the result is as shown below-

```
PES1UG20CS280:Pavan -$nano myls.c
PES1UG20CS280:Pavan -$gcc myls.c -o myls
myls.c: In function 'main':
myls.c:3:2: warning: implicit declaration of function 'system' [-Wimplicit-function-declaration]
    3 |  system("ls");
      |  ^~~~~~
PES1UG20CS280:Pavan -$sudo chown root myls
[sudo] password for seed:
PES1UG20CS280:Pavan -$sudo chmod 4755 myls
PES1UG20CS280:Pavan -$ls -l myls
-rwsr-xr-x 1 root seed 16696 Jan 18 04:28 myls
PES1UG20CS280:Pavan -$./myls
a.out  cap_leak.c  catall.c  child  myenv.c  myls  myls.c  myprintenv.c  parent  setuid  setuidenv.c  sysenv  sysenv.c
PES1UG20CS280:Pavan -$
```

In this part of the task, we remove the existing /bin/sh file and use the ENV. variable to set the PATH to the malicious code present in the IS directory. The PATH value is echoed as shown below-

```
PES1UG20CS280(Pavan) -$sudo chown root myls
PES1UG20CS280(Pavan) -$ sudo chmod 4755 myls
PES1UG20CS280(Pavan) -$ls -l myls
-rwsr-xr-x 1 root seed 7344 Jan 17 23:44 myls
PES1UG20CS280(Pavan) -$./myls
ls  ls.c  myls  myls.c
PES1UG20CS280(Pavan) -$gcc ls.c -o ls
PES1UG20CS280(Pavan) -$sudo rm /bin/sh
PES1UG20CS280(Pavan) -$ sudo ln -sf /bin/zsh /bin/sh
PES1UG20CS280(Pavan) -$export PATH=/home/seed/Desktop/IS:$PATH
PES1UG20CS280(Pavan) -$echo $PATH
/home/seed/Desktop/IS:/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/u
sr/bin:/sbin:/bin:/usr/games:/usr/local/games:.:/usr/bin:/snap/bin:/usr/lib/jvm/
java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/j
re/bin:/home/seed/android/android-sdk-linux/tools:/home/seed/android/android-sdk
-linux/platform-tools:/home/seed/android/android-ndk/android-ndk-r8d:/home/seed/
.local/bin
PES1UG20CS280(Pavan) -$./myls
This is the malicious program!
real uid is 1000
effective uid is 0
PES1UG20CS280(Pavan) -$
```

Once set, the corresponding myls code is re-executed. The resolution for ls is now redirected to the IS directory where the malicious C code ls.c is executed.

The real UID of the ls.c file is seed (created by the SEED user) and therefore it shows the RUID as 1000.

The subsequent program calling this file, myls.c is root-owned and being executed there, therefore, the effective user id is that of the root (0).

INFORMATION SECURITY LAB 01

Name: Pavan R Kashyap SRN: PES1UG20CS280
6th Semester E section

TASK 7 THE LD PRELOAD Environment Variable and Set-UID programs

The LD_PRELOAD variable is used to mention the library(-ies) that will be loaded before anything else during the linking (dynamic linking at run time) stage.

In this task, a malicious code mylib.c is written. This malicious code overrides the sleep() function. This code is compiled and changed into a shared library with the name libmylib.so.1.0.1

Subsequently, after that, the LD_PRELOAD variable's value is assigned to that library. Myprog.c contains the definition of the sleep() function, the implementation of the sleep() function is obtained from the shared libraries that will be linked to that file during runtime.

When myprog.c is executed, the malicious code's sleep() function is included (during the dynamic linking phase), and that is why "I am not sleeping!" is printed on the screen.

```
PES1UG20CS280(Pavan) -$sudo ln -sf /bin/bash /bin/sh
PES1UG20CS280(Pavan) -$PATH=$(getconf PATH)
PES1UG20CS280(Pavan) -$gedit mylib.c
PES1UG20CS280(Pavan) -$gcc -fPIC -g -c mylib.c
PES1UG20CS280(Pavan) -$gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
PES1UG20CS280(Pavan) -$export LD_PRELOAD=./libmylib.so.1.0.1
PES1UG20CS280(Pavan) -$gedit myprog.c
PES1UG20CS280(Pavan) -$gcc myprog.c -o myprog
PES1UG20CS280(Pavan) -$./myprog
I am not sleeping!
```

When the privileges of the executable myprog are escalated and executed, we see that the executable uses the system sleep() function instead of the malicious shared library defined.

This indicates that when we escalate the privilege of the executable to root, the linker links the shared library defined within root (the system sleep() call), defined in the LD_LIBRARYPATH variable.

Therefore, there is no result printed.

```
PES1UG20CS280(Pavan) -$ sudo chown root myprog
PES1UG20CS280(Pavan) -$ sudo chmod 4755 myprog
PES1UG20CS280(Pavan) -$./myprog
```

In this subtask, we add the malicious shared library path while acting as the root user. This indicates that when the linker resolves the library to load/link to the myprog executable, it now uses the malicious shared library defined by the root user itself. Therefore, on linking and loading, the malicious sleep() function overrides the system sleep() function and prints the result displayed below-

```
PES1UG20CS280(Pavan) -$ sudo su
PES1UG20CS280_ROOT(10.0.2.5) -$export LD_PRELOAD=./libmylib.so.1.0.1
PES1UG20CS280_ROOT(10.0.2.5) -$ ./myprog
I am not sleeping!
PES1UG20CS280_ROOT(10.0.2.5) -$exit
exit
PES1UG20CS280(Pavan) -$
```

INFORMATION SECURITY LAB 01

Name: Pavan R Kashyap SRN: PES1UG20CS280
6th Semester E section

In this subtask we create a new user “user1” with details defined correspondingly. Once created we make user1 the owner of the myprogl executable and use the Set-UID program to escalate the privileges of the seed user who intends to execute it.

The LD_PRELOAD variable of the seed user is set to the malicious shared library; the same isn’t done for user1.

Therefore, when the seed user executes the myprogl executable, the linker links the libraries defined in user1 (and not in seed). Since user1 doesn’t have LD_PRELOAD variable set to the malicious shared library, the linker loads/links the system sleep() call. Therefore, the corresponding output is as shown below-

```
PES1UG20CS280(Pavan) -$sudo adduser user1
Adding user `user1' ...
Adding new group `user1' (1001) ...
Adding new user `user1' (1001) with group `user1' ...
Creating home directory `/home/user1' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for user1
Enter the new value, or press ENTER for the default
        Full Name []: James Joseph
        Room Number []: 15C
        Work Phone []: 008900
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
PES1UG20CS280(Pavan) -$gcc myprog.c -o myprog1
PES1UG20CS280(Pavan) -$sudo chown user1 myprog1
PES1UG20CS280(Pavan) -$ sudo chmod 4755 myprog1
PES1UG20CS280(Pavan) -$export LD_PRELOAD=./libmylib.so.1.0.1
PES1UG20CS280(Pavan) -$ ./myprog1
PES1UG20CS280(Pavan) -$
```

INFORMATION SECURITY LAB 01

Name: Pavan R Kashyap SRN: PES1UG20CS280
6th Semester E section

TASK 8 Invoking external programs using system() versus execve() command

The major difference between system() and execve() command is that the execve() command separates the code from the data , whilst system() command does not. This makes the system() command vulnerable to code injections.

Two files are created – one myfile and one rootfile. The root file is moved to root user privilege using the chown command. The catall.c is compiled, and changed into a Set-UID program.

In the first subtask, system() command is used. The user has injected the rm command to the data section. When the system(command) function is called, the call resolves to system(/bin/cat myfile; rm rootfile).

This causes the contents of the myfile to be printed and the contents of the rootfile(which is owned by the root) to be deleted as shown below-

```
PES1UG20CS280(Pavan) -$cat > myfile
HELLO THERE PAVAN HERE!
PES1UG20CS280(Pavan) -$cat > rootfile
ATTACKER'S NAME IS PAVAN!
PES1UG20CS280(Pavan) -$sudo chown root rootfile
PES1UG20CS280(Pavan) -$gcc catall.c -o catall
PES1UG20CS280(Pavan) -$sudo chown root catall
PES1UG20CS280(Pavan) -$sudo chmod 4755 catall
PES1UG20CS280(Pavan) -$./catall "myfile;rm rootfile"
HELLO THERE PAVAN HERE!
PES1UG20CS280(Pavan) -$cat rootfile
cat: rootfile: No such file or directory
PES1UG20CS280(Pavan) -$
```

Even when the seed user does not have privilege to delete root files, through the system() call and privilege escalation, the seed user could delete the rootfile.

Name: Pavan R Kashyap SRN: PES1UG20CS280
6th Semester E section

However, execve() does not operate the same way. Execve() separates the data section from the command section, ensuring that data does not become code. So, when the previous subtask is repeated, we see that the execve() command takes the first argument as /bin/cat (the code argument) and the second argument as myfile;rm rootfile (the data argument). This separation ensures that code injections aren't possible.

The execve() command looks for a file named "myfile;rm rootfile" and finds none. So, it immediately states that no such file or directory exists, thereby saving the rootfile from getting deleted.

```
PES1UG20CS280(Pavan) -$gedit catall.c
PES1UG20CS280(Pavan) -$cat > rootfile
ATTACKER'S NAME IS PAVAN! IT IS A SECRET!
PES1UG20CS280(Pavan) -$sudo chown root rootfile
PES1UG20CS280(Pavan) -$gcc catall.c -o catall
PES1UG20CS280(Pavan) -$sudo chown root catall
PES1UG20CS280(Pavan) -$sudo chmod 4755 catall
PES1UG20CS280(Pavan) -$./catall "myfile;rm rootfile"
/bin/cat: 'myfile;rm rootfile': No such file or directory
PES1UG20CS280(Pavan) -$cat rootfile
ATTACKER'S NAME IS PAVAN! IT IS A SECRET!
PES1UG20CS280(Pavan) -$cat myfile
HELLO THERE PAVAN HERE!
PES1UG20CS280(Pavan) -$
```

INFORMATION SECURITY LAB 01

Name: Pavan R Kashyap SRN: PES1UG20CS280
6th Semester E section

Task 9 Capability leaking

In this task, we understand how important it is to close file descriptors after usage.

Zzz file is created in the /etc directory by the root user. The seed user has permission only to view it, which is why when the cat command is used, the seed user is able to see the contents of the zzz file.

The cap_leak.c code opens the intended file and then downgrades its privileges (cap_leak's privileges) on completion.

The getuid() command in the cap_leak.c code is used to obtain the user id of the seed user (in our case). Subsequently, the setuid() command is used to set the user id of the cap_leak file to seed user (downgrading it from root user to seed user).

When a file is to be read/written/executed, its file privileges are first checked. Once opened, these privileges aren't verified/checked for every read() or write() call that is executed, while the file is still open.

When the privileges of the cap_leak file are downgraded, the file is still open (the file descriptor still holds some value). This indicates that my capleak executable (now in the seed user space) can still access and alter the /etc/zzz file (which is a root privilege file) even though cap_leak's privileges got downgraded.

So once done, a new shell is opened and the file descriptor is retained. The seed user, who previously did not have the permission/capability to alter the contents of the /etc/zzz file , can now alter it easily.

The echo command redirects data into the file and the source file is now altered, because the file was not closed (the fd still held some value).

```
PES1UG20CS280(Pavan) -$su root
Password:
PES1UG20CS280_ROOT(10.0.2.5) -$cat > /etc/zzz
ZZZ PAVAN HERE!
PES1UG20CS280_ROOT(10.0.2.5) -$exit
exit
PES1UG20CS280(Pavan) -$cat /etc/zzz
ZZZ PAVAN HERE!
PES1UG20CS280(Pavan) -$sudo chown root /etc/zzz
PES1UG20CS280(Pavan) -$sudo chmod 0644 /etc/zzz
PES1UG20CS280(Pavan) -$gcc cap_leak.c -o capleak
PES1UG20CS280(Pavan) -$sudo chown root capleak
PES1UG20CS280(Pavan) -$sudo chmod 4755 capleak
PES1UG20CS280(Pavan) -$./capleak
fd is 3
sh-4.3$ echo "malicious data" > &3
sh: syntax error near unexpected token `&'
sh-4.3$ echo "malicious data" >&3
sh-4.3$ cat /etc/zzz
ZZZ PAVAN HERE!
malicious data
sh-4.3$
```