1 Introduction

The learning objective of this assignment is for students to understand how environment variables affect program and system behaviors. Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer. They are used by most operating systems, including Unix and Windows. Although environment variables affect program behaviors, how they achieve that is not well understood by many programmers. Therefore, if a program uses environment variables, but the programmer do not know that they are used, the program may have vulnerabilities.

In this assignment, students will learn how environment variables work, how they are propogated from parent process to child, and how they affect system/program bahivors. We are particularly interested in how environment variables affect the behavior of `SETUID` programs, which are usually privileged programs.

`SETUID` is an important security mechanism in Unix operating systems. When a regular program is run, it runs with the privilege of the user executing that program. When a `SETUID` program is run, it runs with the privilege of the program file owner. For example, if the program's owner is root, then when anyone runs this program, the program gains root's privileges during its execution. `SETUID` allows us to perform essential tasks, such as changing passwords, but vulnerabilities in `SETUID` programs can allow an adversary to perform local privilege escalation.

While the `SETUID` concept is limited to Unix, the problems of dangerous environment variables and local privilege escalation exists on all operating systems. The techniques that we will use to mitigate such vulnerabilities in this assignment can be applied to non-`SETUID` programs to protect those programs as well. This assignment is based on the NSF SEED Lab project directed by Kevin Du and uses the SEED Lab VM.

2 References

1. Matt Bishop. How to Write a Setuid Program, http://nob.cs.ucdavis.edu/bishop/secprog/1987-sproglogin.pdf.
2. Hao Chen, David Wagner, and Drew Dean. Setuid Demystified, http://www.cs.berkeley.edu/~daw/papers/setuid-usenix02.pdf.
3. James Walden. UNIX and C/C++ Resources, http://faculty.cs.nku.edu/~waldenj/unix_and_c.html.

3 Resources Required

In this assignment, students will need to run the SEED Ubuntu Linux VM as a guest operating system inside a virtual machine. Each student has a copy of this VM on the CS vSphere cluster at coivcenter1.hh.nku.edu, which can be accessed via the vSphere web or desktop clients as shown in class.

4 Initial setup

We will need to create some SETUID programs to analyze in this assignment. In your home directory, create a subdirectory named assignment1, in which we will create files and perform the tasks for this assignment. After creating this directory, create SETUID copies of the bash and zsh shells in this directory.

```
$ mkdir assignment1
$ cd assignment1
$ sudo -s
[sudo] password for seed: (enter seed password)
# cp /bin/bash .
# chmod 4755 bash
# cp /bin/zsh .
# chmod 4755 zsh
```

We also need to replace bash with zsh as the default system shell for reasons that will become apparent when we analyze the security of these two shells.

```
# cd /bin
# rm sh
# ln -s zsh sh
# exit
```

In the assignment1 directory, create file named myls.c containing the C program code below.

```
#include <stdlib.h>
int main()
{
    system("ls");
    return 0;
}
```

Compile the program using the following commands, verify that it works by running it to list the files in the current directory, then make it SETUID.

```
$ gcc -o myls myls.c
$ ls
$ ./myls          # output should be identical to that of ls
[sudo] password for seed: (enter seed password)
# chown root myls
# chmod 4755 myls
# exit
```

5 Tasks

In this assignment, you will explore the SETUID mechanism in Linux in a variety of ways, then write a report to describe your discoveries. You are required to accomplish the tasks listed in the subsections below. The first few tasks are simple and will not require extensive writeups in the report. Later tasks require more work and more extensive writeups.

5.1 Environment variables

In this task, we study how environment variables can be set and inherited between programs. We will use the `bash` shell in the default `seed` account. We can print the environment variables with the `printenv` command, but there is too much output to read on one screen so we will page the output using `less`. The key `q` will exit `less`.

```
$ printenv | less
```

We can print a single environment variable by giving its name as an argument to `printenv` or search through the list of environment variables using the `grep` command. Remember that search patterns are case-sensitive regular expressions unless you use the `-i` option to `grep` to make them case-insensitive.

```
$ printenv PATH
$ printenv | grep PATH
```

We can use `export` and `unset` to set or unset environment variables in your current process inside `bash`. It should be noted that these two commands are not seperate programs; they are internal commands and you will not be able to find them outside of `bash`.

```
$ export MYTEST=foo
$ printenv MYTEST
$ unset MYTEST
$ printenv MYTEST
```

In Unix, `fork()` creates a new process by duplicating the calling process. The new process, referred to as the child, is a duplicate of the calling process, referred to as the parent; however, several things are not inherited by the child (please see the manual of `fork()` by typing the following command: `man fork`). In this task, we will explore whether the parent's environment variables are inherited by the child process or not.

Save the following program text from the box below in a file named `myenv.c`. Compile and run the following program, then run the program and observe its output. Because the output is long and we will need it for comparison purposes later, run the program again and save its output to the file `envs.child`.

```
$ gcc -o myenv myenv.c
$ ./myenv
$ ./myenv >envs.child
```

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

extern char **environ;

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

void main()
{
  pid_t childPid;

  switch(childPid = fork()) {
    case 0:  /* child process */
        printenv();
    exit(0);
    default:  /* parent process */
        //printenv();
exit(0);
  }
}
```

Next, comment out the `printenv()` statement in the child process case, and uncomment the `printenv()` statement in the parent process case. Compile and run the code, and describe your observation. Save the output in another file. Compare the difference of these two files using the `diff` command. This command will print no output if the two files given as arguments are identical.

```
$ gcc -o myenv myenv.c
$ ./myenv
$ ./myenv envs.parent
$ diff envs.child envs.parent
```

5.2 `SETUID` Shells

In the normal user account (not as root), run the `id` command, run the `SETUID` copy of `zsh` you created above, then run the `id` command again. Observe what has changed in the privilege level as revealed by the `id` command.

```
$ id
$ ./zsh
$ id
```

```
$ exit
```

Repeat the steps above for `bash` and observe the differences. What has changed and why. You may need to do some online research to determine how and why the differences you observed exist.

5.3 PATH environment variable

The `system(const char *cmd)` library function can be used to execute a command within a program. The way `system(cmd)` works is to invoke the `/bin/sh` program, and then let the shell program to execute `cmd`. More information about `system` can be obtained by reading the manual page via the command `man 3 system`, where 3 is the section of the manual for programmers and `system` is the function of interest.

Because of the shell program invoked, calling `system()` within a `SETUID` program is extremely dangerous. This is because the actual behavior of the shell program can be affected by environment variables, such as `PATH`. These environment variables are under user's control and must be treated as dangerous like any type of input that can be controlled by an attacker.

By changing these variables, malicious users can control the behavior of the `SETUID` program. In `bash`, you can change the `PATH` environment variable in the following way (this example adds the directory `/home/seed` to the beginning of the `PATH` environment variable):

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

The `SETUID` program `myls` created above is supposed to execute the `/bin/ls` command; however, the programmer only uses the relative path for the `ls` command, rather than the absolute path in the code.

1. Can you let this `SETUID` program (owned by root) run your code instead of `/bin/ls`? If you can, is your code running with the root privilege? Describe the setup you had to create to do this, including all commands executed and their output if any, and explain your observations.
2. Now, change `/bin/sh` so it points back to `/bin/bash`, undoing the change we performed during the setup for this assignment, and repeat the above attack. Can you still get the root privilege? Describe and explain your observations.

5.4 The difference between `system()` and `execve()`

Before you work on this task, please make sure that `/bin/sh` *is pointed to* `/bin/zsh`.

Background: Bob works for an auditing agency, and he needs to investigate a company for a suspected fraud. For the investigation purpose, Bob needs to be able to read all the files in the company's `Unix` system; on the other hand, to protect the integrity of the system, Bob should not be able to modify any file.

To achieve this goal, Vince, the superuser of the system, wrote a special `SETUID` root program (see below), and then gave the executable permission to Bob. This program requires Bob to type a file name at the command line, and then it will run `/bin/cat` to display the specified file. Since the program is running as a root, it can display any file Bob specifies. However, since the program has no write operations, Vince is

very sure that Bob cannot use this special program to modify any file. Read and compare the man pages for the `system` and `exec` system calls to understand why Vince is so certain.

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

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

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

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

  /* Set q = 0 for Question 1, and q = 1 for Question 2 */
  int q = 0;
  if (q == 0){
    char *command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
    sprintf(command, "%s %s", v[0], v[1]);
    system(command);
  }
  else execve(v[0], v, 0);

  return 0 ;
}
```

1. Set $q = 0$ in the program. This way, the program will use `system()` to invoke the command. Is this program safe? If you were Bob, can you compromise the integrity of the system? For example, can you remove any file that is not writable to you? (Hint: remember that `system()` actually invokes `/bin/sh`, and then runs the command within the shell environment. We have tried the environment variable in the previous task; here let us try a different attack. Please pay attention to the special characters used in a normal shell environment).
2. Set $q = 1$ in the program. This way, the program will use `execve()` to invoke the command. Do your attacks in task (a) still work? Please describe and explain your observations.

5.5 The LD_PRELOAD environment variable.

To make sure `SETUID` programs are safe from the manipulation of the `LD_PRELOAD` environment variable, the runtime linker (`ld.so`) will ignore this environment variable if the program is a `SETUID` root program, except for some conditions. We will figure out what these conditions are in this task.

1. Let us build a dynamic link library. Create the following program, and name it `mylib.c`. It basically overrides the `sleep()` function in `libc`:

```
#include <stdio.h>
void sleep (int s)
{
printf("I am not sleeping!\n");
}
```

2. We can compile the above program using the following commands (in the `-Wl` argument, the third character is ℓ, not one; in the `-lc` argment, the second character is ℓ):

```
$ gcc -fPIC -g -c mylib.c
$ gcc -shared -Wl,-soname,libmylib.so.1 -o libmylib.so.1.0.1 mylib.o -lc
```

3. Now, set the `LD_PRELOAD` environment variable:

```
$ export LD_PRELOAD=./libmylib.so.1.0.1
```

4. Finally, compile the following program `myprog` (put this program in the same directory as `libmylib.so.1.0.1`):

```
/* myprog.c */
int main()
{
sleep(1);
return 0;
}
```

Please run `myprog` under the following conditions, and observe what happens. Based on your observations, tell us when the runtime linker will ignore the LD_PRELOAD environment variable, and explain why.

- Make `myprog` a regular program, and run it as a normal user.
- Make `myprog` a `SETUID` root program, and run it as a normal user.
- Make `myprog` a `SETUID` root program, and run it in the root account.
- Make `myprog` a `SETUID` user1 program (i.e., the owner is user1, which is another user account), and run it as a different user (not-root user).

5.6 Relinquishing privileges and cleanup.

To improve security by following the Principle of Least Privilege, `SETUID` programs usually call `setuid()` system call to permanently relinquish their root privileges. However, sometimes, this is not enough. Compile the following program, and make the program a `SETUID` root program. Run it in a normal user account, and describe what you have observed. Will the file `/etc/zzz` be modified? Please explain your observation.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

void main() {
  int fd;

  /* Assume that /etc/zzz is an important system file,
   * and it is owned by root with permission 0644.
   * Before running this program, you should creat
   * the file /etc/zzz first. */
  fd = open("/etc/zzz", O_RDWR | O_APPEND);
  if (fd == -1) {
      printf("Cannot open /etc/zzz.\n");
      exit(0);
  }

  /* Simulate the tasks conducted by the program */
  sleep(1);

  /* After the task, the root privileges are no longer needed,
     it's time to relinquish the root privileges permanently. */
  setuid(getuid());  /* getuid() returns the real uid */

  if (fork()) { /* In the parent process */
    close (fd);
    exit(0);
  } else { /* in the child process */
    /* Now, assume that the child process is compromised, malicious
       attackers have injected the following statements
       into this process */

    write (fd, "Malicious Data\n", 15);
    close (fd);
  }
}
```

6 Deliverables

Organize your report with sections that have the numbers and names of the subsections of the Tasks section above. Turn in a hardcopy of your assignment report in class and e-mail to me an electronic attachment named `a1-lastname-firstname.odt` with your actual last name and first place used in place of the placeholders.