**Lab Report #1**

**Name: Nicolas Rios CIS 495 + 595**

Students need to submit a detailed lab report to describe what they have done, what they have observed, and how they interpret the results. Reports should include evidences to support the observations. Evidences include packet traces, screenshots, etc.

[LAB TASKS 1](#_TOC_250001)

1. MANIPULATING ENVIORNMENT VARIABLES 1
2. PASSING ENVIORNMENT VARIBALE FROM PARENT PROCESS TO CHILD PROCESS 4
3. ENVIORNMENT VARIABLES AND EXECVE() 6
4. ENVIORNMENT VARIABLES AND SYSTEM() 8
5. ENVIORNMENT VARIABLES AND SET-UID PROGRAMS 9
6. THE PATH ENVIORNMENT VARIABLE AND SET-UID PROGRAMS 13
7. THE LD\_PRELOAD ENVIORNMENT VARIABLE AND SET-UID PROGRAMS 15
8. INVOKING EXTERNAL PROGRAMS USING SYSTEM() VERSUS EXECVE() 19
9. CAPABILITY LEAKING 21
10. SUMMARY 23

# Lab Tasks

1. Manipulating Environment Variables
   * By utilizing the printenv command I was able to see all of the environment variables currently available to the user. I then utilized the export command to set my own environment variable for the session, TZ. After setting this variable to “America/Chicago”, I used printenv to check if it was created. Following that this, I used the unset command on the TZ environment variable to remove it from the session.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
| *Printenv* command |  |
| *Export* and *unset* commands |  |

1. Passing Environment Variables from Parent Process to Child Process
   * For this task, I first compiled and saved the results of running the myprintenv.c file into the results1 file. This output showed the environment variables for the child process. Next, I edited the myprintenv.c file to show the environment variables for the parent process and saving the results into the results2 file. I then utilized the diff command to compare these two output files and discovered that the two process had almost identical results aside from the GNOME terminal screen and service variables. These variables are used for the terminal emulator. The PWD (path of current directory) variables was also slightly different with the spelling of Labsetup.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
| Step 1 (Child Process Environment Variables) |  |
| Step 2 (Parent Process Environment Variables) |  |
| Step 3 (Comparing using *diff* command) |  |

1. Environment Variables and execve()
   * In this task, by running and compiling the original myenv.c file, the program did not printout any environment variables. Meaning that no environment variables were passed into the new program, since it is being executed within the calling process. However, when we modify the code as in Step 2, by changing NULL to environ, this variable points to the array of environment variables for the environment rather than the new program. This results in the display of the environment variables for the calling process instead of the new program.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
| Step 1 |  |
| Step 2 |  |

1. Environment Variables and system()
   * For this task, I created a new file called mysysenv.c to simulate the use of the system() function of execution of programs rather than execve(). As shown in the screenshot, by using the system() function the shell is being asked to execute the program, which is why the environment variables for the calling process/shell are printed out.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
|  |  |

1. Environment Variables and Set-UID Programs
   * In this task, I created a new program called foo.c that prints out the environment variables of its current shell. I then changed the owner of the file to root as well as its permissions to 4755 ( to make it a Set-UID program). Then, I set the LD\_LIBRARY\_PATH and NRIOS environment variables for the current shell, since the PATH environment variable was already established. I then executed the foo.c program and found that the environment variables I created in the shell were inherited and printed out by the Set-UID program.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
| Step 1 |  |
| Step 2 |  |
| Step 3 |  |

1. The PATH Environment Variable and Set-UID Programs
   * In this task, I altered the PATH environment variable that is inherited by the Set-UID program to list the directory files of the Labsetup folder. However, when I created my own *ls* function to override the one in the Set-UID program, the Linux terminal blocked the compile command to avoid my malicious attack/code.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
| Attempting to manipulate system() function in Set-UID program |  |

1. The LD\_PRELOAD Environment Variable and Set-UID Programs
   * In this task, I set up my own Dynamic Linker and Library to define my own sleep function as shown in the first screenshot. I then created the myprog.c file to test the use of the LD\_PRELOAD Environment Variable and Set-UID Programs under various scenarios. I found that within each scenario the result was the same, the program utilized the sleep function defined in my library. I believe that this was due to the setting of the LD\_PRELOAD environment variable before running the program since this allowed the program to first call from this library when starting and thus executing the modified sleep function.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
| Setting up Dynamic Linker and Library |  |
| Myprog as normal user |  |
| Myprog as Set-UID and normal user |  |
| Myprogram as root Set-UID and LD\_PRELOAD as root environment variable |  |
| Myprog as user1 Set-UID and user1 environment variables |  |

1. Invoking External Programs using system() versus execve()
   * In this task, I compiled and ran the catall program as a Set-UID program. In the first try, I utilized the system function to execute commands directly from the shell. However, the problem with this is that the program reads the user input as data for commands. Thus, I was able to delete the test.c file from the directory utilizing the catall program. In the second attempt, I utilized the execve function in catall.c instead, and found that the same attack did not work and the entire user input was treated as a string.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
| System() |  |
| Execve() |  |

1. Capability Leaking
   * In this task, I created a file in the /etc directory called zzz by using the root account. I then returned to the seed account to compile and run the cap\_leak.c file as a Set-UID program. However, once I ran this program it gave me the file descriptor of the zzz file along with root access. I then utilized this file descriptor to alter the zzz file as the seed user since this file was not properly closed after downgrading the privileges of the program.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
| Cap\_Leak Vulnerability |  |

1. Summary
   * Overall, this lab showed me how to utilize various commands within the Linux terminal and feel comfortable navigating and completing necessary tasks within this environment. Additionally, by exploring the vulnerabilities associated with environment variables and how they influence the capabilities of child processes as well as shells. Task 5 greatly aided me in understanding this concept. Moreover, I got to see how capability leaking can occur and be utilized by attackers to still utilize root privileges even after there EUID is set their RUID.