Environment Variable and Set-UID Program Lab

JIAYU LI

Overview

- Introduction
- Manipulating Environment Variables
- Passing Environment Variables from Parent Process to Child Process
- Environment Variables with execve() and system()
- The PATH, LD_PRELOAD Environment Variable and Set-UID Programs
- Capability leaking

Introduction

- Understanding how **environment variables** work, how they are **propagated from parent process to child**, and how they **affect system/program behaviors**.
- Exploring how **environment variables** affect the behavior of **Set-UID programs**.

In this week, the lab covers the following topics:

- Environment variables
- Set-UID programs
- Dynamic loader/linker
- Securely invoke external programs
- Capability leaking

Manipulating Environment Variables

What you need to know in this part is that:

Some **commands** can be used to set and unset environment variables.

- ***printenv*** or ***env*** command to print out the environment variables.

- For obtaining specific environment variables like *"**PWD**",* we have *"**printenv PWD**"* or *"**env | grep PWD**".*

- Use ***export*** and ***unset*** to set or unset environment variables

Passing Environment Variables from Parent Process to Child Process

- Unix creates a new process by duplicating the calling process using fork() .
- When creating a child process from its parent, something is not inherited by the child
- We study whether the parent's **environment variables** are inherited by the child process or not.
- **Note**: Using files in **Labsetup** folder and knowing the commands about manipulating environment variables

Environment Variables with execve() and system()

- Exploring how environment variables are affected when a new program is executed via ***execve()*** and ***system()***.
- ***execve()*** calls a system call to load a new command and execute it.
  - *int execve(const char *filename, char *const argv[], char *const envp[])*
  - **Hint: are the environment variables automatically inherited by a new program?**
- ***system()*** executes "/bin/sh -c command"and asks the shell to execute the command

- *execve("/usr/bin/env", argv, NULL);*
- *execve("/usr/bin/env", argv, environ);*
- *system("/usr/bin/env");*
- What are differences among these three commands?

The PATH Environment Variable and Set-UID Programs

- A **Set-UID program** runs, it assumes the owner's privileges.
- **Root** is the owner of a **Set-UID program**, **anyone** running the program can gains the **root's privileges** during its execution.
- Users can indeed affect the **behaviors of a Set-UID program** via **environment variables**.

The PATH Environment Variable and Set-UID Programs

CAUTION:

- How to change a program's ownership to root and make it a **Set-UID program**.
- **/bin/sh** is actually a symbolic link pointing to **/bin/dash** iUbuntu 20.04. This shell program has a countermeasure that prevents itself from being executed in a Set-UID process.
- We have installed a shell program called **zsh** in our Ubuntu 20.04 VM. You can follow the command to link **/bin/sh** to **/bin/zsh**: ***$ sudo ln -sf /bin/zsh /bin/sh***

The PATH Environment Variable and Set-UID Programs

- Calling **system()** within a **Set-UID program** is quite dangerous because the actual behavior of the shell program can be affected by **environment variables**, such as **PATH**.

- Changing the **PATH environment variable** in Bash: ***$ export PATH=/home/seed:$PATH***

- Thinking: changing the owner of the left program as root and making it a Set-UID program.
- Question: 1. Can you let this Set-UID program run your **ls** instead of **/bin/ls**?

  2. Is your code running with the **root privilege**?

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

The LD_PRELOAD Environment Variable and Set-UID Programs

- *LD_PRELOAD, LD_LIBRARY PATH* and other *LD ** influence the behavior of dynamic loader/linker. **A dynamic loader/linker** is the part of an operating system that loads and links the shared libraries needed by an executable at run time.

- *LD_PRELOAD* is an environmental variable containing one or more paths to shared libraries, or shared objects, that the loader will load before any other shared library including the C runtime library (libc.so). Preloading a library means that its functions will be used before others of the same name in later libraries.

- For example, you implements alternative *malloc* function. By preloading the new library using *LD_PRELOAD* the new *malloc* functions will be used rather than the corresponding standard libc functions.

- **Think** about a normal user with a *Set-UID* root program when exporting/unset the customized *LD_PRELOAD* environmental variable; **Think** about real UID and effective UID in the scenario.

Set-UID program with system()

- **system()** function is used to execute a command.

- **system()** actually executes ***"/bin/sh -c command"***, i.e., it executes ***/bin/sh***, and asks the shell to execute the command.

- If you look at the implementation of the **system()** function, you will see that it uses **execl()** to execute ***/bin/sh***; **execl()** calls **execve()**, passing to it the environment variables array. Therefore, using **system()**, the environment variables of the calling process is passed to the new program ***/bin/sh***.

- Think about if a normal user runs a Set-UID root program with **system(input)**.
  - For example, if the input is “/bin/cat ” + “something which a normal user types”
  - What will happened?

Capability Leaking

- Set-UID programs often permanently relinquish their root privileges if such privileges are not needed anymore. Therefore, root privileges must be revoked.
- The ***setuid()*** system call revoke the privileges. ***setuid()*** sets the effective user ID of the calling process.
  - *If the effective UID of the caller is root, the real UID and saved set-user-ID are also set. Therefore, if a Set-UID program with effective UID 0 calls setuid(n), the process will become a normal process, with all its UIDs being set to n.*
- When revoking the privilege, one of the common mistakes is capability leaking.
- The process may have gained some privileged capabilities when it was still privileged; when the privilege is downgraded, if the program does not clean up those capabilities, they may still be accessible by the non-privileged process.
  - In other words, although the effective user ID of the process becomes non-privileged, the process is still privileged because it possesses privileged capabilities.

Reference

1. https://seedsecuritylabs.org/Labs_20.04/Software/Environment_Variable_and_SetUID/