环境变量与 Set-UID 程序实验

1 概述

本实验的学习目地是让学生理解环境变量如何对程序和系统产生影响。环境变量是一组动态的命名值，可以影响计算机上进程的行为。自 1979 年被引入 Unix 系统以来，环境变量被大多数操作系统使用。尽管环境变量会影响程序行为，但许多程序员对其工作原理并不熟悉。因此，如果一个程序使用了环境变量，但程序员对此并不了解，那么该程序可能存在漏洞。

在本实验中，学生将学习环境变量的工作原理、它们如何从父进程传入到子进程，以及它们如何影响系统/程序的行为。我们尤其关注环境变量如何影响 `Set-UID` 程序的行为，这些程序通常是特权程序。本实验涵盖以下主题：

- 环境变量
- `Set-UID` 程序
- 安全调用外部程序
- Capability 泄露

阅读资料和视频。 有关 `Set-UID` 机制、环境变量及其相关安全问题的详细内容可以参考以下资料：

- SEED 书的第 1 和第 2 章，*Computer & Internet Security: A Hands-on Approach*, 3rd Edition, by Wenliang Du. 详情请见 `https://www.handsonsecurity.net`.
- SEED 视频的第 2 节，*Computer Security: A Hands-on Approach*, by Wenliang Du. 详情请见 `https://www.handsonsecurity.net/video.html`.

实验环境。 本实验在 SEED Ubuntu 20.04 VM 中测试可行。您可以从 SEED 网站上下载我们预先构建好的镜像并在您自己的电脑上运行 SEED VM。然而，大多数 SEED 实验可以在云端进行，您可以按照我们的说明在云端创建 SEED VM。

2 实验任务

实验所需的文件包含在 `Labsetup.zip` 中，可以从实验网站下载。

2.1 任务 1: 操作环境变量

在此任务中，我们学习可以用来设置和取消环境变量的命令。我们使用 SEED 账户中的 Bash。用户使用的默认 shell 在 `/etc/passwd` 文件中设置（每条记录的最后一个字段）。您可以使用命令 `chsh` 将其更改为其他 shell 程序（请勿在本实验中更改）。请完成以下任务：

- 使用 printenv 或 env 命令打印出环境变量。如果您对某些特定的环境变量感兴趣，例如 PWD，可以使用 "printenv PWD" 或 "env | grep PWD"。
- 使用 export 和 unset 设置或取消环境变量。需要注意的是，这两个命令不是独立的程序，而是 Bash 的内部命令（您无法在 Bash 之外找到它们）。

2.2 任务 2: 从父进程向子进程传递环境变量

在此任务中，我们研究子进程如何从其父进程获取环境变量。在 Unix 系统中，fork() 通过复制调用进程来创建新进程。新进程（称为子进程）是调用进程（称为父进程）的副本。然而，有些内容并不会被子进程继承（请通过命令 man fork 查看 fork() 的手册）。在此任务中，我们想了解父进程的环境变量是否会被子进程继承。

步骤 1. 请编译并运行以下程序，并描述您的观察结果。该程序可以在 Labsetup 文件夹中找到；可以使用 "gcc myprintenv.c" 编译，生成的二进制文件名为 a.out。运行它并使用 "a.out > file" 将输出保存到文件中。

Listing 1: myprintenv.c

```
#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:  /* 子进程 */
      printenv();          ①
      exit(0);
    default:  /* 父进程 */
      //printenv();        ②
      exit(0);
  }
}
```

步骤 2. 注释掉子进程情况中的 printenv() 语句（第 ① 行），并取消注释父进程情况中的 printenv() 语句（第 ② 行）。再次编译并运行代码，并描述您的观察结果。将输出保存到另一个文件中。

步骤 3. 使用 diff 命令比较这两个文件的差异。请得出您的结论。

2.3 任务 3：环境变量与 execve()

在本任务中，我们研究如果使用 execve() 执行新程序，环境变量会受到怎样的影响。execve() 函数会调用系统调用来加载并执行新的程序。该函数不会创建新的进程，而是会用新的程序的内容覆盖进程的代码和数据区域，也就是说，execve() 是在调用进程的内部运行新程序。我们关注的是环境变量的变化，它们是否会被保留在新程序中?

步骤 1. 请编译并运行以下程序，并描述你的观察结果。此程序简单地执行名为 /usr/bin/env 的程序，该程序会打印当前进程的环境变量。

Listing 2: myenv.c

```
#include <unistd.h>

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

  argv[0] = "/usr/bin/env";
  argv[1] = NULL;
  execve("/usr/bin/env", argv, NULL);    ①

  return 0 ;
}
```

步骤 2. 将第 ① 行中的 execve() 调用改为以下形式，描述你的观察结果。

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

步骤 3. 请总结你的结论，说明新程序如何获得其环境变量。

2.4 任务 4：环境变量与 system()

在本任务中,我们研究如果通过 system() 函数来执行新程序,环境变量会受到什么影响。system() 函数用于执行一个命令，但不同于直接执行命令的 execve()，system() 实际执行的是 "/bin/sh -c command"，即它会先执行 /bin/sh，然后让 shell 来执行命令。

如果查看 system() 函数的实现，可以发现它使用 execl() 来执行 /bin/sh，execl() 会调用 execve()，同时将环境变量传递给这个系统调用。因此，使用 system() 时，调用进程的环境变量会传递给新程序 /bin/sh。请编译并运行以下程序以验证这一点。

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

int main()
{
  system("/usr/bin/env");
  return 0 ;
}
```

2.5 任务 5：环境变量与 Set-UID 程序

Set-UID 是 Unix 操作系统中一个重要的安全机制。当一个 Set-UID 程序运行时，它会获得文件所有者的权限。例如，如果程序的所有者是 root，那么任何人运行这个程序时，该程序在运行期间都会获得 root 权限。Set-UID 机制让我们实现了许多有用的功能，但由于它会提升用户权限，因此风险较大。尽管 Set-UID 程序的行为由其程序逻辑决定，不是由用户决定，但用户可以通过环境变量影响程序行为。为了理解 Set-UID 程序如何受到环境变量影响，我们首先要弄清楚 Set-UID 程序的进程是否会从用户的进程中继承环境变量。

步骤 1. 编写以下程序，用于打印当前进程中的所有环境变量。

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

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

步骤 2. 编译上述程序，将其所有者改为 root，并将其设置为 Set-UID 程序。

```
// 假设程序名为 foo
$ sudo chown root foo
$ sudo chmod 4755 foo
```

步骤 3. 在你的 shell 中（需要是普通用户账户，而不是 root 账户），使用 export 命令设置以下环境变量（它们可能已经存在）：

- PATH
- LD_LIBRARY_PATH
- ANY_NAME （这是由你定义的环境变量，可以选择任意名称）

这些环境变量会在用户的 shell 进程中设置。现在，在你的 shell 中运行步骤 2 中的 Set-UID 程序。当你在 shell 中输入程序名时，shell 会 fork 一个子进程，并使用子进程运行该程序。请检查你在 shell 进程（父进程）中设置的所有环境变量是否都进入了 Set-UID 子进程。描述你的观察结果。如果有让你感到意外的地方，请描述它们。

2.6 任务 6：PATH 环境变量与 Set-UID 程序

由于 system() 会调用 shell 程序，在 Set-UID 程序中调用 system() 是非常危险的。这是因为 shell 程序的行为可能受到环境变量（如 PATH）的影响。这些环境变量是由用户提供的，而用户是不可信的。通过更改这些变量，恶意用户可以控制 Set-UID 程序的行为。在 Bash 中，你可以通过以下方式更改 PATH 环境变量（此示例将目录 /home/seed 添加到 PATH 环境变量的开头）：

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

下面的 Set-UID 程序应该执行 /bin/ls 命令，然而，程序员只使用了相对路径来执行 ls 命令，而不是绝对路径：

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

请编译上述程序，将其所有者改为 root，并将其设置为 Set-UID 程序。你能否让这个 Set-UID 程序运行你自己的恶意代码，而不是 /bin/ls？如果可以，你的恶意代码是否以 root 权限运行？描述并解释你的观察结果。

注意： system(cmd) 函数会首先执行 /bin/sh 程序，然后让该 shell 程序运行 cmd 命令。在 Ubuntu 20.04（以及之前的几个版本）中，/bin/sh 实际上是一个指向 /bin/dash 的符号链接。该 shell 程序有一种防范措施，可以防止自己在 Set-UID 进程中被执行。如果 dash 检测到它在 Set-UID 进程中执行，它会立即将有效用户 ID 更改为进程的真实用户 ID，从而放弃特权。

由于我们的受害程序是一个 Set-UID 程序，/bin/dash 中的对策可以阻止我们的攻击。为了查看没有这种对策的情况下攻击的效果，我们将 /bin/sh 链接到另一个没有此防范措施的 shell，我们已在 Ubuntu 20.04 虚拟机中安装了一个这样的名为 zsh 的 shell 程序。我们使用以下命令将 /bin/sh 链接到 /bin/zsh：

```
$ sudo ln -sf /bin/zsh /bin/sh
```

2.7 任务 7: LD_PRELOAD 环境变量与 Set-UID 程序

在本任务中,我们研究 Set-UID 程序如何处理一些环境变量。包括 LD_PRELOAD、LD_LIBRARY_PATH 以及其他 LD_* 在内的多个环境变量会影响动态加载器/链接器的行为。动态加载器/链接器是操作系统的一部分，它负责将共享库加载到内存并进行链接。

在 Linux 系统中，ld.so 或 ld-linux.so 是动态加载器/链接器（每个针对不同类型的二进制文件)。在影响其行为的环境变量中，LD_LIBRARY_PATH 和 LD_PRELOAD 是本实验关注的两个变量。加载器/链接器首先需要找到程序需要的共享库，在查找标准目录之前会首先搜索 LD_LIBRARY_PATH 环境变量里指定的目录。如果 LD_PRELOAD 环境变量存在，它里面指定了用户定义的一组共享库，这些库会在所有其他库之前被加载。本任务仅研究 LD_PRELOAD。

步骤 1. 首先，我们观察这些环境变量在运行普通程序时如何影响动态加载器/链接器的行为。请按照以下步骤操作:

1. 构建一个动态链接库。创建以下程序，命名为 mylib.c。该程序覆盖了 libc 中的 sleep() 函数:

```
#include <stdio.h>
void sleep (int s)
{
  /* 如果这是由特权程序调用的,
     你可以在这里造成破坏!  */
  printf("我不会睡觉! \n");
}
```

2. 使用以下命令编译上述程序（-lc 参数中的第二个字符是 ℓ):

```
$ gcc -fPIC -g -c mylib.c
$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
```

3. 设置 LD_PRELOAD 环境变量:

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

4. 最后，编译以下程序 myprog，并放置在与上述动态链接库 libmylib.so.1.0.1 相同的目录中:

```
/* myprog.c */
#include <unistd.h>
int main()
{
  sleep(1);
  return 0;
}
```

步骤 2. 完成上述操作后，在以下条件下运行 myprog，观察发生的情况。

- 将 myprog 设置为普通程序，并以普通用户运行。
- 将 myprog 设置为 Set-UID root 程序，并以普通用户运行。
- 将 myprog 设置为 Set-UID root 程序，在 root 账户中设置 LD_PRELOAD 环境变量后运行该程序。
- 将 myprog 设置为 Set-UID user1 程序（即拥有者为 user1，这是另一个用户账户），在 user2 账户（非 root 用户）中设置 LD_PRELOAD 环境变量后运行该程序。

步骤 3. 在上述场景中，即使运行的是相同的程序，您应该会观察到不同的行为。您需要弄清楚导致差异的原因。环境变量在这里起到了作用。请设计实验以确定主要原因，并解释为何步骤 2 中的行为会有所不同。（提示：子进程可能不会继承 LD_* 环境变量）。

2.8 任务 8: 使用 system() 和 execve() 调用外部程序

虽然 system() 和 execve() 都可以用来运行新程序，但如果在特权程序（例如 Set-UID 程序）中使用 system()，则可能非常危险。我们已经看到 system() 如何受 PATH 环境变量的影响，因为该变量会影响 shell 的行为。而 execve() 没有这个问题，因为它不会调用 shell。调用 shell 还有另一个危险的后果，这次与环境变量无关。让我们看以下场景。

Bob 在一家审计机构工作，他需要调查一家公司是否存在欺诈行为。为了调查，Bob 需要能够读取公司 Unix 系统中的所有文件。但是为了保护系统的完整性，Bob 不应该能够修改任何文件。为此，系统的超级用户 Vince 编写了一个特殊的 set-root-uid 程序（见下文），并授予 Bob 可执行权限。该程序要求 Bob 在命令行输入一个文件名，然后它将运行 /bin/cat 来显示指定的文件。由于程序以 root 权限运行，它可以显示 Bob 指定的任何文件。然而，由于该程序没有写的操作，Vince 确信 Bob 无法使用这个特殊程序修改任何文件。

Listing 3: catall.c

```
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]);

  // 仅使用以下两种方法之一。
```

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

  return 0 ;
}
```

步骤 1. 编译上述程序，使其成为一个 root 拥有的 Set-UID 程序。程序将使用 system() 调用命令。如果你是 Bob，你能破坏系统的完整性吗？例如，你是否可以删除系统里的任何文件？

步骤 2. 注释掉 system(command) 语句，取消注释 execve() 语句，程序将使用 execve() 来运行命令。编译程序并使其成为一个 root 拥有的 Set-UID 程序。在步骤 1 中的攻击是否仍然有效？请描述并解释你的观察。

2.9 任务 9: 权限泄漏

为了遵循最小权限原则，Set-UID 程序通常会在不再需要权限时永久放弃其 root 权限。此外，有时程序需要将控制权移交给用户，所以在交权之前，Set-UID 程序会放弃 root 权限。我们可以使用 setuid() 系统调用来撤销权限。setuid() 设置调用进程的有效用户 ID，但如果调用者的有效 UID 为 root，则实际 UID 和保存的 set-user-ID 也会被设置。因此，如果具有有效 UID=0 的 Set-UID 程序调用 setuid(n)，则进程将成为普通进程，其所有 UID 都将被设置为 n。

在撤销权限时，一个常见的错误是权限泄漏。进程在仍具有权限时可能已经获得了一些特权功能。当权限被降级时，如果程序没有清除这些功能，它们可能仍然可以被非特权进程访问。换句话说，尽管进程的有效用户 ID 已变为非特权用户，但进程仍可能拥有特权。

编译以下程序，将其所有者更改为 root，并使其成为一个 Set-UID 程序。以普通用户身份运行该程序。你能利用该程序中的权限泄漏漏洞吗？你的目标是做到能够以普通用户的身份修改 /etc/zzz 文件。

Listing 4: cap_leak.c

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

  /* 假设 /etc/zzz 是一个重要的系统文件，
   * 并且它的所有者是 root，权限是 0644。
   * 在运行此程序之前，你应先创建 /etc/zzz 文件。 */
  fd = open("/etc/zzz", O_RDWR | O_APPEND);
  if (fd == -1) {
     printf("Cannot open /etc/zzz\n");
     exit(0);
  }
```

```
  // 打印文件描述符值
  printf("fd is %d\n", fd);

  // 通过将有效 uid 设置为与实际 uid 相同来永久放弃特权
  setuid(getuid());

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

3 提交

你需要提交一份带有截图的详细实验报告来描述你所做的工作和你观察到的现象。你还需要对一些有趣或令人惊讶的观察结果进行解释。请同时列出重要的代码段并附上解释。只是简单地附上代码不加以解释不会获得学分。