环境变量和设置-UID 程序实验室

1 概述

本实验室的学习目标是让学生了解环境变量如何影响程序和系统行为。 环境变量是一组动态命名值，它可以影响运行过程在计算机上的行为方式。它们被大多数操作系统使用，因为它们在 1979 年被引入 Unix。虽然环境变量会影响程序行为，但许多程序员并不清楚它们是如何实现的。 作为结果，如果程序使用环境变量，但程序员不知道它们被使用，程序可能会有漏洞。 在这个实验室中，学生将了解环境变量是如何工作的，它们是如何从父母过程到孩子的，以及它们如何影响系统/程序的 bahories。我们特别感兴趣的是环境变量如何影响 Set-UID 程序的行为，这些程序通常是特权程序..

2 实验室任务

2.1 任务 1：管理环境变量

在这个任务中，我们研究可以用来设置和取消环境变量的命令。 我们在种子帐户中使用 Bash。 用户使用的默认 shell 设置在/etc/passwd 文件中（每个条目的最后一个字段）。 您可以使用 chsh 命令将此更改为另一个 shell 程序（请不要为本实验室这样做）。 请你完成以下任务：.

- 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，您可以使用“printenvPWD”或“env|grepPWD”。
- 使用 export 和 unset 设置或 unset 环境变量.. 应该注意的是，这两个命令不是单独的程序；它们是 Bash 的两个内部命令(您将无法在 Bash 之外找到它们)。

2.2 任务 2：继承来自父母的环境变量

在这个任务中，我们研究环境变量是如何由孩子从他们的父母那里继承的。 在 Unix 中，fork()通过复制调用进程来创建一个新进程。 新进程，简称子进程，是调用进程的精确重复，简称父进程；但是，有几个东西不是子进程继承的(请参阅 fork 手册()通过键入以下命令：manfork).. 在这个任务中，我们想知道父进程的环境变量是否由子进程继承。

步骤 1 请编译并运行以下程序，并描述您的观察。 因为输出包含多个字符串，所以应该将输出保存到一个文件中，例如使用 a.out>子文件(假设 a.out 是可执行文件名)..

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

```
extern char**en；

无效 Printenv()
{
    int i=0；
    而(环境[i]！=零){printf(“%s\n”，环境[i]；我++；
    }
}

无效的主要()
{
    pid_t 孩子皮德；

    开关(子 PID=叉（）){情况 0： /*儿童进程*/
            printenv()；
            退出（0）；
       默认情况：/*父进程*/printenv()；
            退出（0）；
    }
}
```

步骤 2 现在，在子进程案例中注释掉 printenv()语句，并在父进程案例中取消注释 printenv()语句。 编译并运行代码，并描述您的观察。 将输出保存在另一个文件中。

步骤 3 使用 diff 命令比较这两个文件的差异.. 请得出你的结论。

2.3 任务 3：环境变量和执行()

在这个任务中，我们研究了当一个新的程序通过 execve()执行时，环境变量是如何受到影响的.. 函数 execve()调用系统调用来加载新命令并执行它；此函数永远不会返回。 没有创建新进程；相反，调用进程的文本、数据、bss 和堆栈被加载的程序覆盖。 本质上，execve()在调用过程中运行新程序。 我们对环境变量会发生什么感兴趣；它们是由新程序自动继承的吗？

步骤 1 请编译并运行以下程序，并描述您的观察。 此程序只需执行一个名为/usr/bin/env 的程序，该程序打印出当前进程的环境变量。

```
#包括<stdio.h>
#包括<stdlib.h>

extern char**en；

主要()
{
    char*argv[2]；
```

```
    argv[0]="/usr/bin/env";
    阿格夫[1]=零；

    execve("/usr/bin/env"，argv，NULL)；

    返回 0；
}
```

步骤 2 现在，将 execve()的调用更改为以下内容，并描述您的观察。

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

步骤 3 请就新程序如何获得其环境变量得出您的结论。

2.4 任务 4：环境变量和系统()

在本任务中，我们研究了当一个新的程序通过系统()函数执行时，环境变量是如何受到影响的.. 此函数用于执行命令，但与直接执行命令的 execve（)不同，系统实际上(）执行"/bin/sh-c 命令"，即它执行/bin/sh，并要求 shell 执行该命令。

如果查看系统()函数的实现，会看到它使用 execl()执行/bin/sh；excel()调用 execve()，传递给它环境变量数组.. 因此，使用系统()，调用进程的环境变量传递给新程序/bin/sh。 请编译并运行以下程序来验证这一点。

```
#包括<stdio.h>
#包括<stdlib.h>

主要()
{
    系统("/usr/bin/env")；

    返回 0；
}
```

2.5 任务 5：环境变量和集合-UID 程序

集合-UID 是 Unix 操作系统中重要的安全机制。 当 SET-UID 程序运行时，它假定所有者的特权。 例如，如果程序的所有者是 root，那么当任何人运行此程序时，程序在执行过程中会获得 root 的特权。 Set-UID 允许我们做许多有趣的事情，但是它在执行时会升级用户的特权，这使得它非常危险。 虽然 SET-UID 程序的行为是由其程序逻辑决定的，而不是由用户决定的，但用户确实可以通过环境变量来影响这些行为。 为了了解 Set-UID 程序是如何受到影响的，让我们首先弄清楚 Set-UID 程序的进程是否从用户的进程继承了环境变量。

步骤 1 我们将编写一个程序，可以打印出当前进程中的所有环境变量。

```
#包括<stdio.h>
#包括<stdlib.h>

extern char**en；
```

```
无效的主要()
{
    int i=0；
    而(环境[i]！=零){printf(“%s\n”，环境[i]；我++；
    }
}
```

步骤 2 编译上述程序，将其所有权更改为 root，并使其成为 Set-UID 程序..

步骤 3 在您的 Bashshell 中（需要在普通用户帐户中，而不是根帐户中），使用 export 命令设置以下环境变量（它们可能已经存在）：.

- 路径
- ld_library_path
- ANY_NAME（这是由您定义的环境变量，所以选择您想要的任何名称）。

在用户的 shell 进程中设置这些环境变量.. 现在，从 shell 中的步骤 2 运行 Set-UID 程序。 在您的 shell 中键入程序名称后，shell 将分叉一个子进程，并使用子进程运行程序。 请检查在 shell 进程（父进程）中设置的所有环境变量是否进入 Set-UID 子进程。 描述你的观察。 如果你有惊喜，描述一下。

2.6 任务 6：PATH 环境变量和 SET-UID 程序

由于调用了 shell 程序，在 Set-UID 程序中调用系统()是非常危险的。 这是因为 shell 程序的实际行为可能会受到环境变量的影响，例如 PATH；这些环境变量是由用户提供的，用户可能是恶意的。 通过改变这些变量，恶意用户可以控制 SET-UID 程序的行为.. 在 Bash 中，可以通过以下方式更改 PATH 环境变量(本例将目录 /home/seed 添加到 PATH 环境变量的开头)：.

```
$出口 PATH=/家庭/种子：$PATH
```

下面的 Set-UID 程序应该执行/bin/ls 命令；但是，程序员只使用 ls 命令的相对路径，而不是绝对路径：

```
主要()
{
        系统(“ls”)；返回 0；
}
```

请编译上述程序，并将其所有者更改为 root，并使其成为 Set-UID 程序。 您能让这个 SET-UID 程序运行您的代码而不是/bin/ls 吗？ 如果可以，您的代码是否使用根特权运行？ 描述并解释你的观察。

2.7 任务 7：LD_PRELOAD 环境变量和 Set-UID 程序

在这个任务中，我们研究了 Set-UID 程序如何处理一些环境变量。 几个环境变量，包括 LD_PRELOAD、LD_LIBRARY_PATH 和其他 LD_*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(OS)的一部分，它加载(从持久存储到 RAM)并在运行时链接可执行文件所需的共享库。

在 Linux 中，ld.so 或 ld-linux.so 是动态加载器/链接器（每个都用于不同类型的二进制文件）。在影响他们行为的环境变量中，LD_LIBRARY_PATH 和 LD_PRELOAD 是我们在本实验室中所设想的两个变量。在 Linux 中，LD_LIBRARY_PATH 是一组冒号分隔的目录，在标准目录集之前，应该首先搜索库。 LD_PRELOAD 指定要在所有其他库之前加载的附加的、用户指定的共享库的列表。 在这个任务中，我们只会研究 LD_PRELOAD。

步骤 1 首先，我们将看到这些环境变量在运行正常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：.

1. 让我们建立一个动态链接库。 创建以下程序，并将其命名为 mylib.c。 基本覆盖 libc：中的睡眠()功能.

```
#包括<stdio.h>
空虚睡眠(INT)
{
    /*如果这是由特权程序调用的，您可以在这里进行损坏！ */printf(“我没睡！\n”);
}
```

2. 我们可以使用以下命令编译上面的程序(在-lc 分段中，第二个字符是 Z)：

```
gcc-f PIC-g-c mylib.c
%gcc-shared-o libmylib.so.1.0.1mylib.o-lc
```

3. 现在，设置 LD_PRELOAD 环境变量：.

```
出口 LD_PRELOAD=/libmylib.so.1
```

4. 最后，编译下面的程序 myprog，并将其放在与上面的动态链接库相同的目录 libmylib.so.1.0.1：.

```
/*myprog.c*/
主要()
{
   睡眠（1）；
   返回 0；
}
```

步骤 2 在您完成上述操作后，请在以下条件下运行 myprog，并观察发生了什么。

- 使 myprog 成为常规程序，并作为普通用户运行..
- 使 myprog 成为 SET-UID 根程序，并作为普通用户运行..
- 使 myprog 成为 Set-UID 根程序，在根帐户中再次导出 LD_PRELOAD 环境变量并运行它。

- 使 myprog 成为 Set-UIDuser1 程序(即所有者是 user1，这是另一个用户帐户)，在不同的用户帐户（非根用户）中再次导出 LD_PRELOAD 环境变量并运行它。

步骤 3 您应该能够在上面描述的场景中观察不同的行为，即使您正在运行相同的程序。 你需要弄清楚是什么导致了差异。 环境变量在这里发挥作用.. 请设计一个实验，找出主要原因，并解释为什么步骤 2 中的行为是不同的。 (提示：子进程可能不继承 LD_*环境变量)。

2.8 任务 8：使用系统（)和执行(）调用外部程序

虽然系统()和 execve（)都可以用来运行新的程序，但如果在特权程序中使用系统(），如 Set-UID 程序，则是相当危险的。我们已经看到了 PATH 环境变量如何影响系统()的行为，因为变量会影响 shell 的工作方式.. 执行()没有问题，因为它不调用 shell。 调用 shell 有另一个危险的后果，这一次，它与环境变量无关。 让我们看看下面的场景。

鲍勃在一家审计机构工作，他需要调查一家公司涉嫌欺诈。 为了调查目的，Bob 需要能够读取公司 Unix 系统中的所有文件；另一方面，为了保护系统的完整性，Bob 不应该能够修改任何文件。 为了实现这一目标，系统的超级用户 Vince 编写了一个特殊的 set-root-uid 程序（见下文），然后将可执行权限授予 Bob。此程序要求 Bob 在命令行中键入文件名，然后运行/bin/cat 来显示指定的文件。 由于程序作为根目录运行，它可以显示 Bob 指定的任何文件。 然而，由于程序没有写操作，文斯非常确信 Bob 不能使用这个特殊程序来修改任何文件。

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

主(intargc，char*argv[])
{
    char*v[3]；
    char*command；

    如果(argc<2){printf(“请键入文件名.\n”)；返回 1；
    }

    v[0]=“/bin/cat”；v[1]=argv[1]；v[2]=NULL；

    命令=malloc(strlen(v[0])+strlen(v[1])；sprintf(命令，“%s%s”，v[0]，v[1])；

    //只使用以下一个。 系统（命令）；
    //执行(v[0]，v，NULL)；

    返回 0；
}
```

步骤 1：编译上述程序，使其所有者根目录，并将其更改为 SET-UID 程序。程序将使用系统()调用该命令。如果你是鲍勃，你能损害系统的完整性吗？ 例如，您能删除对您不可写的文件吗？

步骤 2：注释出系统（命令）语句，并取消注释 execve()语句；程序将使用 execve()调用该命令。 编译程序，并使其 Set-UID(由 root 拥有)。 步骤 1 中的攻击仍然有效吗？ 请描述并解释你的观察。

2.9 任务 9：能力泄漏

为了遵循最小特权原则，如果不再需要这些特权，Set-UID 程序通常会永久地放弃其根本特权。此外，有时程序需要将其控制移交给用户；在这种情况下，必须撤销根特权。 可以使用 setuid()系统调用来撤销特权。 根据手册，“setuid()设置调用过程的有效用户 ID。 如果调用方的有效 UID 是 root，则还将设置真正的 UID 和保存的 set-user-ID“。 因此，如果具有有效 UID0 的 SET-UID 程序调用 setuid(N)，则该进程将成为一个正常进程，其所有 UID 都被设置为 n。

当取消特权时，常见的错误之一是能力泄漏。当进程仍然处于特权状态时，可能已经获得了一些特权功能；当特权被降级时，如果程序不清理这些功能，它们仍然可以被非特权进程访问。 换句话说，虽然进程的有效用户 ID 变得非特权，但进程仍然具有特权，因为它具有特权功能。

编译下面的程序，将其所有者更改为 root，并使其成为 Set-UID 程序。 作为正常用户运行程序，并描述您所观察到的内容。 文件/etc/zzz 是否会被修改？ 请解释你的观察。

```
#包括在内    <stdio.h>
#包括在内    <stdlib.h>
#包括在内    <fcntl.h>空主(){intfd;

    /*   假设        该/etc/zzz 是一个重要的系统文件，由 root 拥有权限 0644。运行此程
    *    还有它      序，您应该创建
         以前                                          */
    *    首先是文件/etc/zzz。
    fd   =打开(“/etc/zz”，O_RDWR|O_APPEND)；
    如   (fd==-1){printf(“无法打开/etc/zz\n”)；exit（0）；
    果



    /*睡眠模拟程序执行的任务*/



    /*在任务之后，不再需要根特权，是时候放弃 setuid(getuid（）)了；/* ()
                                                  根特权永久。 * 返回真正的 uid*/



    如果（叉()）{/* 在父关闭(FD)；                    进程*/
```

```
        退出（0）；
    }子进程中的其他{/*
        /*现在，假设子进程被破坏，恶意攻击者已将以下语句注入此进程*/

        写(FD，“恶意数据\n”，15)；
        关闭(FD)；
    }
}
```

3 提交

你需要提交一份详细的实验室报告来描述你所做的和你观察到的事情，包括截图和代码片段。 您还需要对有趣或令人惊讶的观察提供解释。您被鼓励进行进一步的调查，超出实验室描述的要求。你可以获得额外努力的加分（由你的教练决定）。