Laboratorio de Variables de Entorno y Programas `Set-UID`

1 Descripción General

El objetivo de este laboratorio es que los estudiantes entiendan como las variables de entorno afectan el comportamiento de un programa y el del sistema. Las variables de entornos son un conjunto de valores dinámicos con un nombre que pueden afectar la forma en que se comportan los procesos que se encuentran corriendo. Son usadas por la mayoría de los sistemas operativos, desde su introducción en Unix en 1979. Aunque las variables de entorno tienen estos impactos en los programas, muchas veces no son del todo comprendidas por los programadores. Por lo tanto si estas variables son usadas y el programador no sabe que estan siendo usadas, el programa puede tener potenciales vulnerabilidades.

En este laboratorio, los estudiantes entenderán como funcionan las variables de entorno, como son propagadas desde un proceso padre a un proceso hijo y como esto afecta al comportamiento de los programas y al sistema. Nos centraremos particularmente en programas `Set-UID`, los programas `Set-UID` usualmente son programas privilegiados.

Este laboratorio cubre los siguientes tópicos:

- Variables de Entorno
- Programas `Set-UID`
- Invocar de forma segura programas externos
- Loader/Linker Dinámico
- Fugas/Leaking

Lecturas y Videos. Para una cobertura más detallada sobre el mecanismo `Set-UID`, variables de entorno y sus problemas de seguriad puede consultar:

- Capítulos 1 y 2 del libro de SEED, *Computer & Internet Security: A Hands-on Approach*, 2nd Edition, by Wenliang Du. Para más detalles `https://www.handsonsecurity.net`.
- Sección 2 del curso de SEED en Udemy, *Computer Security: A Hands-on Approach*, by Wenliang Du. Para más detalles `https://www.handsonsecurity.net/video.html`.

Entorno de Laboratorio. Este laboratorio ha sido testeado en nuestra imagen pre-compilada de una VM con Ubuntu 20.04, que puede ser descargada del sitio oficial de SEED. Sin embargo, la mayoría de nuestros laboratorios pueden ser realizados en la nube para esto Ud. puede leer nuestra guía que explica como crear una VM de SEED en la nube.

2 Tareas del Laboratorio

Los archivos necesarios para este laboratorio están incluídos en `Labsetup.zip`, este archivo puede ser descargado del sitio oficial del laboratorio.

2.1 Tarea 1: Manipulando Variables de Entorno

En esta tarea, estudiaremos los comandos que pueden ser usados para setear y borrar variables de entorno. Usaremos Bash en la cuenta de seed. La shell por defecto que tiene este usuario esta configurada en el archivo `/etc/passwd`. Puede usar otra shell usando el comando `chsh` (por favor no lo haga para este laboratorio). Por favor realize las siguientes tareas:

- Use el comando `printenv` o `env` para imprimir en pantalla las variables de entorno. Si está interesado en alguna en particular, como `PWD`, puede usar "`printenv PWD`" o "`env | grep PWD`".
- Use `export` y `unset` para setear o borrar variables de entorno. Estos comandos, son comandos internos de la shell Bash por lo que no estarán disponibles si usa otro tipo de shell.

2.2 Tarea 2: Pasando Variables de Entorno de un Proceso Padre a un Proceso Hijo

En esta tarea, estudiaremos como un proceso hijo recibe las variables de entorno desde su proceso padre. En Unix existe una llamada al sistema llamada `fork()` que se encarga de crear nuevos procesos. El nuevo proceso, conocido como proceso hijo es una copia exacta del proceso que invoca a la llamada al sistema `fork()` este proceso invocador es conocido como el proceso padre; aunque el proceso hijo sea una copia exacta del proceso padre, hay muchas cosas que no son heredadas por el hijo (para más información sobre esto consulte el manual de `fork()` tipeando el siguiente comando: `man fork`). En esta tarea, queremos saber si variables de entorno del padre son heredadas por el hijo o no.

Paso 1. Por favor compile el programa, ejecútelo y describa sus observaciones. El programa esta dentro del directorio `Labsetup`; puede ser compilado usando el comando "`gcc myprintenv.c`", el cual generará un binario llamado `a.out`. Ejecútelo y salve su salida en un archivo usando "`a.out > archivo`".

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:  /* child process */
      printenv();          ①
      exit(0);
    default:  /* parent process */
```

```
      //printenv();          ②
      exit(0);
  }
}
```

Paso 2. Comente `printenv()` en el case del proceso hijo (Línea ①), descomente `printenv()` en el case del proceso padre (Línea ②). Compile y ejecute el programa nuevamente y describa sus observaciones. Salve la salida del programa en otro archivo.

Paso 3. Compare la diferencia de los dos archivos usando el comando `diff`. Explique sus conclusiones.

2.3 Tarea 3: Variables de Entorno y `execve()`

En esta tarea, estudiaremos como las variables de entorno son afectadas cuando se ejecuta un nuevo programa a través de `execve()`. La función `execve()` realiza una llamada al sistema que carga un comando y lo ejecuta; esta función no retorna del llamado. No crea un nuevo proceso; en vez de eso, llama a la sección text del proceso, y las secciones data, bss y el stack son sobreescritas por ese programa que se carga. Esencialmente, `execve()` corre un nuevo programa dentro del proceso que lo invoca. Estamos interesados en las variables de entorno; ¿Son heredadas por el nuevo programa?

Paso 1. Por favor compile y ejecute el siguiente programa, describa sus observaciones. Este programa ejecuta un programa llamado `/usr/bin/env`, que muestra las variables de entorno del proceso en curso.

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 ;
}
```

Paso 2. Cambie la invocación a la función `execve()` (Línea ①) con la siguiente declaración; describa sus observaciones.

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

Paso 3. Por favor explique como es que el nuevo programa obtiene sus variables de entorno.

2.4 Tarea 4: Variables de entorno y `system()`

En esta tarea, estudiaremos como las variables de entorno son afectadas cuando se ejecuta un nuevo programa a través de la función `system()`. Esta función se usa para ejecutar un comando pero a diferencia de `execve()` que directamente ejecuta el comando `system()` ejecuta `"/bin/sh -c comando"` es decir ejecuta el comando a través de la invocación de `/bin/sh`.

Si observa la implementación de la función `system(()(`, podrá ver que usa `execl()` para ejecutar `/bin/sh`; `execl()` llama a `execve()`, pasándole un arreglo con las variables de entorno. Ademas usando `system()`, las variables de entorno son pasadas del proceso llamador al nuevo programa `/bin/sh`. Por favor compile y corra el siguiente programa para verificarlo.

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

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

2.5 Tarea 5: Variables de Entorno y Programas `Set-UID`

El bit de `Set-UID` es un mecanismo de seguridad importante dentro de los sistemas operativos basados en UNIX. Cuando se corre un programa con el bit `Set-UID` activado, este asume que los privilegios del programa son los privilegios del propietario del archivo. Por ejemplo, si el propietario del programa es el usuario root, cuando cualquier usuario corra ese programa, el programa tendrá privilegios de root durante la ejecución del mismo. El bit de `Set-UID` nos permite hace varias cosas interesantes, dado que usa los privilegios del usuario propietario para escalar, es algo peligroso. Sin embargo el comportamiento de un programa `Set-UID` está determinado por la lógica del mismo no por los usuarios, los usuarios pueden afectar este comportamiento por medio de las variables de entorno. Para entender como un programa `Set-UID` se ve afectado por esto, vamos a ver como las variables de entorno son heredadas por los procesos de los programas `Set-UID` desde el proceso que corre un usuario.

Paso 1. Escriba el siguiente programa y imprima toda las variables de entorno del proceso en ejecución

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

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

Paso 2. Compile el siguiente programa y haga que el usuario root sea el propietario del mismo y active el bit Set-UID en el programa.

```
// Asssume the program's name is foo
$ sudo chown root foo
$ sudo chmod 4755 foo
```

Paso 3. Dentro de su shell como un usuario normal, use el comando export para setear las siguientes variables de entorno (puede que estás ya existan).

- PATH
- LD_LIBRARY_PATH
- ANY_NAME (Esta variable puede tener cualquier nombre definido por usted).

Estas variables de entorno son seteadas en la shell del usuario que va a correr el proceso. Proceda a corrar el programa Set-UID del Paso 2 en su shell. Después de esto escriba el nombre del programa en su shell, y la shell creará un proceso hijo, use ese proceso hijo para correr el programa. Por favor verifique que todas las variables de entorno seteadas anteriormente en su proceso shell (osea proceso padre) estén dentro del proceso Set-UID hijo. Describa sus observaciones. Si hay algo que le llama atención, no dude en mencionarlo.

2.6 Tarea 6: La Variable de Entorno PATH y Programas Set-UID

Invocar la shell usando system() dentro de un programa Set-UID es algo peligroso. Esto se debe a que el comportamiento de la shell puede ser afectado por las variables de entorno, como PATH; dichas variables de entorno son provistas por el usuario y pueden ser maliciosas. Cambiando estas variables, usuarios maliciosos pueden controlar el comportamiento del programa Set-UID. En Bash, puede cambiar cambiar la variable de entorno PATH de la siguiente manera (En este ejemplo se agrega el directorio /home/seed al principio de la variable de entorno PATH):

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

El programa Set-UID anterior debería de ejecutar el comando /bin/ls; sin embargo, el programador solamente usa el path relativo para el comando /bin/ls en vez del path absoluto:

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

Por favor compile el programa, haga que el usuario root sea el propietario del mismo y active el bit de Set-UID en el programa. ¿Puede hacer que el programa Set-UID corra código malicioso propio en vez de /bin/ls? Si puede hacerlo, ¿Este código se corre con privilegios de root? Describa y explique sus observaciones.

Nota: La función system(cmd) ejecuta primero el programa /bin/sh y después le pide a esta shell que corra el comando cmd. En Ubuntu 20.04 (y versiones anteriores), /bin/sh es un link simbólico que apunta a /bin/dash. Esta shell tiene una protección que impide ser ejecutada en un proceso Set-UID.

Básicamente, si `dash` detecta que esta siendo ejecutada dentro de un proceso `Set-UID`, inmediatamente cambia el effective user ID al real user ID del proceso, eliminando el privilegio.

Dado que nuestro programa objetivo es `Set-UID`, la protección anteriormente mencionada puede frustrar nuestro ataque. Para ver como nuestro ataque funciona sin dicha protección, haremos un link simbólico de la shell `/bin/sh` a otra que no tenga la protección ya mencionada. En nuestra Máquina Virtual de Ubuntu 20.04, hemos instalado una shell llamada `zsh`. Usaremos el siguiente comando para realizar el link de `/bin/sh` a `/bin/zsh`:

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

2.7 Tarea 7: La variable de entorno `LD_PRELOAD` y el Programa `Set-UID`

En esta tarea, estudiaremos como los programas `Set-UID` interactuan con algunas de las variables de entorno. Muchas de estas variables, incluyendo `LD_PRELOAD`, `LD_LIBRARY_PATH`, y otras `LD_*` tienen la capacidad de influenciar en el comportamiento del loader/linker dinámico. El loader/linker dinámico es la parte del sistema operativo que carga (desde un medio de almacenamiento persistente a la RAM) y linkea las librerías compartidas que son necesarias para correr el ejecutable.

En linux, el loader/linker dinámico se encuentra en `ld.so` o `ld-linux.so`. Dentro de las variable de entorno que afectan su comportamiento, nos centraremos en dos que son de especial interés para nuestro laboratorio, `LD_LIBRARY_PATH` y `LD_PRELOAD`. En linux, `LD_LIBRARY_PATH` es un conjunto de directorios separados por dos puntos donde se deben de buscar estas librerías en primera instancia, antes de otros directorios standards del sistema operativo. `LD_PRELOAD` especifíca una lista de librerías compartidas adicionales que son establecidas por el usuario y que deben ser cargadas antes que cualquier otra. En esta tarea sólo estudiaremos `LD_PRELOAD`.

Paso 1. Primero veremos como estas variables de entorno influencian el comportamiento dinámico del loader/linker cuando se corre un programa común y corriente. Siga los siguientes pasos:

1. Vamos a construir una librería dinámica. Cree el siguiente programa y nombrelo como `mylib.c`. La función del mismo es sobreescribir la función `sleep()` en `libc`:

```
#include <stdio.h>
void sleep (int s)
{
  /* If this is invoked by a privileged program,
     you can do damages here!  */
  printf("I am not sleeping!\n");
}
```

2. Podemos compilar el programa usando los siguientes comandos (en el parámetro `-lc` el segundo caracter es ℓ):

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

3. Ahora establezca el valor de la variable de entorno `LD_PRELOAD`:

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

4. Por último compile el siguiente programa `myprog` dentro del mismo directorio en que se encuentra la librería dinámica `libmylib.so.1.0.1` generada en el paso anterior:

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

Paso 2. Después de haber hecho el Paso 1, por favor proceda a ejecutar `myprog` bajo las siguientes condiciones y observe que sucede.

- Haga que `myprog` sea un programa normal y corrálo como un usuario normal.
- Haga que `myprog` sea un programa root `Set-UID` y corrálo como un usuario normal.
- Haga que `myprog` sea un programa root `Set-UID`, exporte la variable de entorno `LD_PRELOAD` en la cuenta root y corrálo.
- Haga que `myprog` sea un programa `Set-UID` cuyo propietario sea user1, exporte la variable de entorno `LD_PRELOAD` nuevamente pero esta vez use una cuenta de usuario diferente que no sea la cuenta de root y corrálo.

Paso 3. Aunque esté corriendo el mismo programa, ud. debería de observar diferentes comportamientos en los escenarios descriptos anteriormente. Necesita descubrir las diferencias y el porque de los resultados entre las diferentes ejecuciones. Las variables de entorno juegan un rol esencial en este escenario. Por favor diseñe un experimento para descubrir las principales causas y explique porque los comportamientos del Paso 2 son diferentes entre sí (Pista: el proceso hijo puede no heredar las variables de entorno `LD_*`).

2.8 Tarea 8: Invocando programas externos, `system()` vs `execve()`

Aunque `system()` y `execve()` pueden ser usados para ejecutar nuevos programas, `system()` es algo peligroso si es usado para ejecutar programas con privilegios, como lo son los programas `Set-UID`. Hemos visto como la variable de entorno PATH afecta el comportamiento de `system()`, porque altera el comportamiento del shell. En cambio `execve()` no tiene este problema, porque no invoca una shell. Invocar una shell tiene otra consecuencia peligrosa y esta vez no tiene nada que ver con las variables de entorno. Observemos los siguientes escenarios.

Bob trabaja para una agencia de auditoría y necesita investigar un caso de posible fraude en un compania. Para esta investigación, Bob necesita leer todos los archivos de la compania que se encuentran en un sistema `Unix`; por otro lado, para proteger la integridad del sistema, Bob no debería de poder modificar ninguno de estos archivos. Para asegurar este cumplimiento, Vince el superusuario del sistema, ha escrito un programa especial set-root-uid (vea más abajo), y le ha dado permisos de ejecución a Bob. Este programa requiere que Bob escriba el nombre del archivo en la línea de comandos y el programa ejecutará `/bin/cat` para mostrar el contenido de este archivo. Dado que el programa está corriendo como root, puede mostrar el contenido de cualquier archivo que Bob escriba. Sin embargo, dado que no tiene permisos de escritura, Vince está seguro que Bob no puede usar este programa para modificar ningún archivo.

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]);

  // Use only one of the followings.
  system(command);
  // execve(v[0], v, NULL);

  return 0 ;
}
```

Paso 1: Compile el programa y haga que el usuario root sea el propietario del mismo y active el bit `Set-UID` sobre el missmo. Este programa usará `system()` para invocar el comando. Si ud. fuera Bob, ¿Podría comprometer la integridad del sistema? Por ejemplo ¿Puede borrar un archivo en el cual no posee permisos de escritura?

Paso 2: Comente el código en la línea `system(command)` y descomente la línea que llama a `execve()`; el programa usará `execve()` para ejecutar el coomando. Compile el programa nuevamente y haga que el usuario root sea el propietario y active el bit `Set-UID`. ¿Funcionan los ataques del Paso 1? Por favor describa y explique sus observaciones.

2.9 Tarea 9: Fuga/Leaking

Siguiendo el principio del Menor Privilegio, los programas `Set-UID` evitan sus privilegios de root si estos no son necesarios. Más aún, a veces los programas necesitan entregar el control al usuario; en este caso, los privilegios de root deben de ser revocados. La llamada al sistema `setuid()` puede ser usada para revocar tales privilegios. De acuerdo al manual "`setuid()` establece el valor effect user ID del proceso que es llamado. Si el effective user ID de quién invoca el programa es el usuario root, está llamada al sistema permite setear estos UIDs". Si un programa `Set-UID` con effective user ID 0 llama a `setuid(n)`, el proceso se convertirá en un proceso normal seteando todos sus UIDs a `n`.

Cuando se revocan privilegios, uno de los errores más comunes es la posibilidad de una fuga o leaking. El proceso pudo obtener capacidades privilegiadas cuando aún estaba corriendo como proceso con privilegios elevados; pero si al momento de revocar esos privilegios no se limpian esas capacidades del programa, estos privilegios pueden permanecer accesibles por el proceso no privilegiado. En otras palabras, aunque el effective user ID del proceso se convierta en un effective user ID no privilegiado, el proceso aún sigue siendo privilegiado porque posee capacidades privilegiadas.

Compile el siguiente programa, haga que el propietario sea el usuario root y active el bit `Set-UID` en el programa. Ejecútelo como un usuario normal. ¿Puede explotar esta capacidad privilegiada que da lugar a

una fuga en el programa? El objetivo es escribir en el archivo `/etc/zzz` siendo un usuario normal.

Listing 4: `cap_leak.c`

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

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

  // Print out the file descriptor value
  printf("fd is %d\n", fd);

  // Permanently disable the privilege by making the
  // effective uid the same as the real uid
  setuid(getuid());

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

3 Informe del Laboratorio

Debe enviar un informe de laboratorio detallado, con capturas de pantalla, para describir lo que ha hecho y lo que ha observado. También debe proporcionar una explicación a las observaciones que sean interesantes o sorprendentes. Enumere también los fragmentos de código más importantes seguidos de una explicación. No recibirán créditos aquellos fragmentos de códigos que no sean explicados.

Agradecimientos

Este documento ha sido traducido al Español por Facundo Fontana