Taller de *syscalls* y señales

Sistemas Operativos

21 de marzo de 2019

Primer cuatrimestre de 2019

1. Antes de empezar

Este taller se podrá resolver utilizando máquinas con sistema operativo Ubuntu. Si lo desean, pueden utilizar la máquina virtual brindada por la materia (`https://campus.exactas.uba.ar/course/view.php?id=1490&section=9`).

2. Ejercicios

2.1. Ejercicio 1

Como parte de un proyecto de ingeniería reversa, se cuenta con un archivo ejecutable (`hai`) y se desea saber qué interacción tiene con el sistema operativo. El ejecutable `hai` requiere que se le pase por parámetro un comando. Un ejemplo sería:

```
./hai echo "Sin ciencia y universidad pública no hay futuro."
```

Se debe analizar el comportamiento del ejecutable mediante la herramienta `strace`, utilizando los argumentos necesarios. No es legal decompilar el programa para entender su comportamiento. Luego, escribir un programa en C que presente el *mismo* comportamiento que `hai`.[1]

2.2. Ejercicio 2 (opcional)

Se pide un programa[2] `justiciero` en C que ejecute el comando pasado por parámetro, no permitiéndole a este enviar una señal a otro proceso mediante `kill`. En caso que el comando especificado intente enviar la señal, se debe abortar su ejecución y mostrar el mensaje: "`Se ha hecho justicia!`".

Por ejemplo:

```
$ ./justiciero kill -9 28988
Se ha hecho justicia!
```

[1] Sugerencia: Usar el programa `hai.c` como base.
[2] Sugerencia: Usar el programa `justiciero.c` como base.

3. Notas útiles para la resolución del taller

3.1. strace

`strace` es una herramienta que permite generar una traza legible de las llamadas al sistema usadas por un programa dado. Sintaxis:

```
$ strace [opciones] comando [argumentos]
```

Algunas opciones útiles:

- `-q`: Omite algunos mensajes innecesarios.
- `-o <archivo>`: Redirige la salida a `<archivo>`.
- `-f`: Traza también a los procesos hijos del proceso trazado.

3.2. ptrace

La *syscall* `ptrace()` permite observar y controlar un proceso hijo. En particular permite obtener una traza del proceso, desde el punto de vista del sistema operativo, al permitir detener el proceso hijo antes y después de realizar un *syscall*.

Su sintaxis es la siguiente:

```
long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);
```

El parámetro `request` permite elegir qué se desea hacer. Dependiendo de este parámetro, algunos de los siguientes parámetros de la *syscall* no se utilizan. Por ejemplo, `PTRACE_TRACEME` no utiliza ninguno de los siguientes tres parámetros, y `PTRACE_POKEDATA` usa todos ellos. `request` puede ser alguno de los siguientes valores:

- `PTRACE_TRACEME`, `PTRACE_ATTACH`, `PTRACE_DETACH`,
- `PTRACE_KILL`, `PTRACE_CONT`,
- `PTRACE_SYSCALL`, `PTRACE_SINGLESTEP`,
- `PTRACE_PEEKDATA`, `PTRACE_POKEDATA`,
- `PTRACE_PEEKUSER`, `PTRACE_POKEUSER`,
- ...y más[3].

El parámetro `pid` es el *process id* del proceso hijo.

3.2.1. PTRACE_SYSCALL

Cada vez que se genera un evento en el proceso hijo, el mismo es detenido. Para continuar la ejecución del proceso hijo se debe hacer una llamada a `ptrace` desde el padre. Esta llamada puede hacerse a `PTRACE_SYSCALL`, `PTRACE_CONT`, o `PTRACE_SINGLESTEP`, dependiendo de qué tipo de evento es el próximo evento que se desea atrapar. Para detenerse por el siguiente ingreso o egreso de una *syscall* se debe usar el valor `PTRACE_SYSCALL`.

3.2.2. PTRACE_KILL

Una forma de terminar el proceso hijo que está siendo monitoreado es enviarle una señal de KILL a través de `ptrace`. Para ello se debe usar el valor de `request` `PTRACE_KILL` e indicar el *pid* del hijo que se desea terminar.

[3] Ver `man 2 ptrace`

3.2.3. PTRACE_PEEKUSER y PTRACE_PEEKDATA

Los request PTRACE_PEEKUSER y PTRACE_PEEKDATA le permiten al proceso padre obtener información sobre la memoria del proceso hijo.

Con PTRACE_PEEKDATA se puede leer *cualquier* dirección del espacio de direcciones del proceso hijo. Pero, aún así, eso no es suficiente, dado que además de los datos visibles desde el proceso hijo, hay más información relativa a este proceso.

Para ello, PTRACE_PEEKUSER nos permite acceder al espacio de memoria *del kernel* que guarda información sobre el proceso hijo. Esta información no es directamente visible desde el proceso hijo, es decir, no está en ninguna dirección de memoria del mismo.

De esta información del kernel, un valor que nos interesa es qué valor tenía el registro EAX al momento de hacer la llamada al sistema, dado que ese valor determina qué *syscall* se está llamando. En el archivo <sys/reg.h> se encuentran definidas algunas constantes útiles, como ORIG_EAX. Dentro de este espacio, el valor de EAX al generarse la llamada al sistema se encuentra en la dirección 4 * ORIG_EAX.

Para hacer una llamada a PTRACE_PEEKUSER o a PTRACE_PEEKDATA la dirección se debe colocar en el parámetro addr, pero el parámetro data no se utiliza. Por el contrario, siempre se lee una *palabra* (4 bytes en el caso de x86) y se devuelven como valor de retorno de la función.

Ejemplo tomado de las slides de la clase:

```
int sysno = ptrace(PTRACE_PEEKUSER, child, 4*ORIG_EAX, NULL);
```

Al utilizar PTRACE_SYSCALL, el proceso se detiene al *entrar* y *salir* de una *syscall*. Para determinar esto, se puede consultar el valor de ptrace(PTRACE_PEEKUSER, child, 4 * EAX, NULL), que devolverá un error (-ENOSYS) cuando el proceso se encuentre entrando en la *syscall*.

3.3. *Includes* recomendados

```
#include <sys/ptrace.h>
#include <sys/wait.h>
#include <sys/reg.h>
#include <unistd.h>
#include <syscall.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
```

3.4. Otros

En los headers <sys/syscall.h> se encuentran definidos símbolos para cada una de las *syscalls* del sistema. Por ejemplo, el número de *syscall* de write está definido por el símbolo SYS_write.