Apunte para la resolución del taller de syscalls

Sistemas operativos

2 cuat. - 2010

1. Enunciado

Ejercicio 1:
Como parte de un proyecto de ingeniería reversa, se cuenta con un archivo ejecutable misterioso (`mister`, en la página de la materia) y se desea saber qué interacción tiene con el sistema operativo. Se debe entregar una breve explicación del mismo.

Ejercicio 2:
Se pide un programa[1] en C que ejecute el comando pasado por parámetro, no permitiéndole a este ejecutar ninguna de las *syscalls prohibidas*: `fork`, `clone`. En caso que el comando especificado intente ejecutar alguna de las *syscalls prohibidas* se debe abortar su ejecución y mostrar el mensaje: "`Syscall prohibida`".

Ejemplo:

```
$ ./nofork date
Wed Sep 11 12:45:13 ART 1985
$ ./nofork time date
Syscall prohibida
```

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` permie elegir qué se desea hacer. Dependiendo de este parámetro, algunos de los siguiente 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`

[1] Sugerencia: Use el programa presentado en clase, `launch.c + ptrace` (slide 23)

- PTRACE_PEEKDATA, PTRACE_POKEDATA
- PTRACE_PEEKUSER, PTRACE_POKEUSER
- ...y más[2]

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

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`.

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.

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 proceso hijo.

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 este caso) 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);
```

3. *Includes* recomendados

```
#include <sys/ptrace.h>
#include <sys/wait.h>
```

[2] Vea `man 2 ptrace`

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

4. Otros

En los headers 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`.