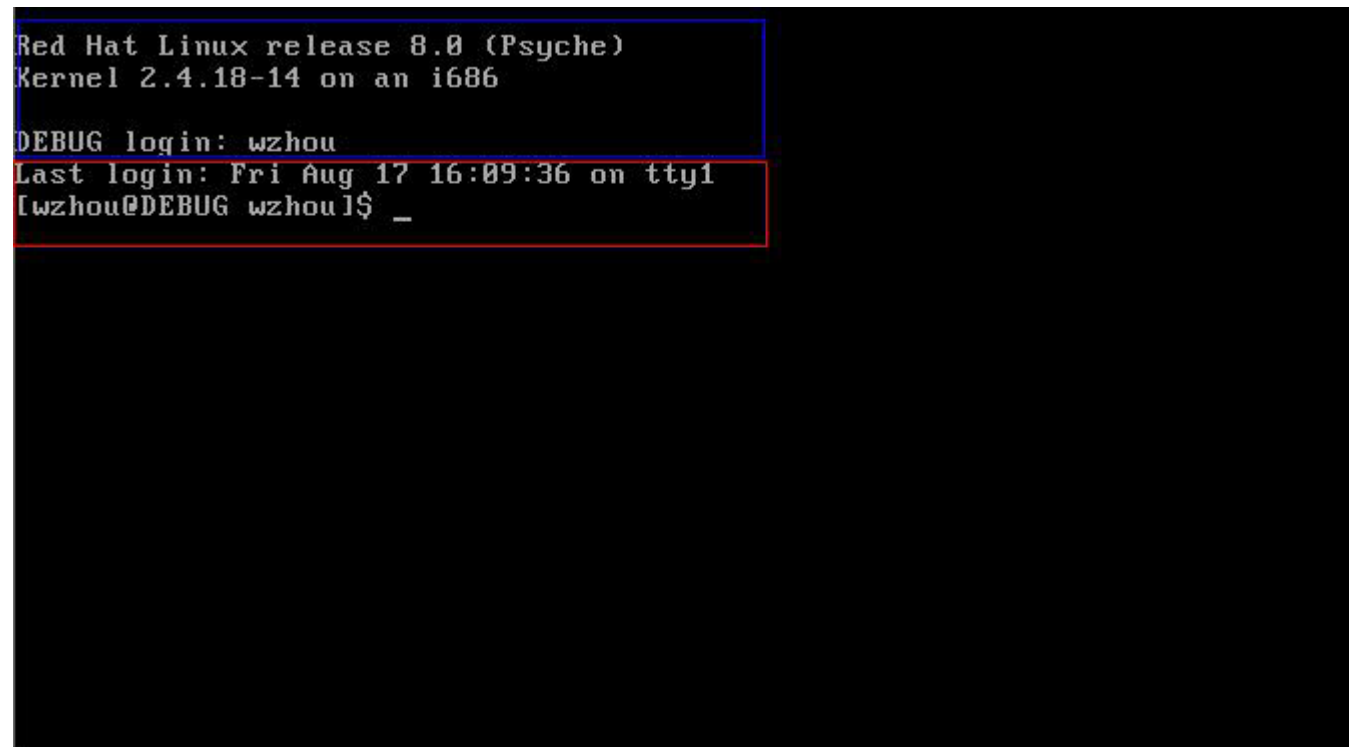


登录(login)程序分析笔记

- 前言 2
- login手册 5
- login分析 13
- login源码 14
- 环境 72
- 联系 73

前言

当用户在登录界面上输入用户名后，login 程序就会起来用密码来验证该用户名的合法性，接着就是一系列的与该用户相关的初始化。

A terminal window with a black background and white text. The text is divided into two sections by a horizontal line. The top section, enclosed in a blue rectangular box, contains the output of the 'mingetty' program: 'Red Hat Linux release 8.0 (Psyche)' and 'Kernel 2.4.18-14 on an i686'. The bottom section, enclosed in a red rectangular box, contains the output of the 'login' program: 'DEBUG login: wzhou', 'Last login: Fri Aug 17 16:09:36 on tty1', and the prompt '[wzhou@DEBUG wzhou]\$ _'.

```
Red Hat Linux release 8.0 (Psyche)
Kernel 2.4.18-14 on an i686

DEBUG login: wzhou
Last login: Fri Aug 17 16:09:36 on tty1
[wzhou@DEBUG wzhou]$ _
```

上图中用蓝线圈起来的是 mingetty 的输出，而红线圈起来的是 login 的输出。Mingetty 接受用户输入的用户名，然后把任务就交给了 login(注意验证用户的密码是 login 的事)。

下面是 mingetty.c 中与此相关的代码。

```
while ((logname = get_logname ()) == 0);  
execl (_PATH_LOGIN, _PATH_LOGIN, "--", logname, NULL);  
error ("%s: can't exec " _PATH_LOGIN ": %s", tty, sys_errlist[errno]);
```

```
while ((logname = get_logname ()) == 0);
```

接受用户输入登录名，logname 即指向获得的登录名。

```
execl (_PATH_LOGIN, _PATH_LOGIN, "--", logname, NULL);
```

这里的_PATH_LOGIN 即是/bin/login，传递给它登录名 logname。

OK，接下来就是 login 的事了。

login process 伴随着系统的整个运行期间，随时恭候着用户的登录。

```
[wzhou@dcmp10 ~]$ ps aux | grep login  
root      29464  0.0  0.0  4392 1032 pts/6    Ss   Sep04   0:00 login -h 13.187.243.55 -p  
root      10669  0.0  0.1  3016 1220 pts/12    Ss   Sep05   0:00 login -h 13.187.243.55 -p  
root      27727  0.0  0.1  4156 1168 pts/17    Ss   Sep07   0:00 login -h 13.187.243.55 -p  
root      27763  0.0  0.0  3540 1032 pts/18    Ss   Sep07   0:00 login -h 13.187.243.55 -p  
root      20979  0.0  0.1  3360 1232 pts/10    Ss   Sep17   0:00 login -h 13.187.243.55 -p  
root       2686  0.0  0.1  2864 1228 pts/3     Ss   Sep25   0:00 login -h 13.187.243.54 -p  
root       7064  0.0  0.1  2564 1232 pts/7     Ss   Sep25   0:00 login -h 13.187.243.55 -p  
wzhou     11309  0.0  0.0  4992  672 pts/2     S+   13:49   0:00 grep login  
[wzhou@dcmp10 ~]$
```

本笔记就是分析 login 到底干了点什么。

login 手册

NAME

login - sign on

SYNOPSIS

```
login [ name ]  
login -p  
login -h hostname  
login -f name
```

DESCRIPTION

login is used when signing onto a system. It can also be used to switch from one user to another at any time (most modern shells have support for this feature built into them, however).

If an argument is not given, login prompts for the username.

If the user is not root, and if /etc/nologin exists, the contents of this file are printed to the screen, and the login is terminated. This is typically used to prevent logins when the system is being taken down.

If special access restrictions are specified for the user in /etc/usertty, these must be met, or the log in attempt will be denied and a syslog message will be generated. See the section on "Special Access Restrictions".

If the user is root, then the login must be occurring on a tty listed in /etc/securetty. Failures will be logged with the syslog facility.

After these conditions have been checked, the password will be requested and checked (if a password is required for this username). Ten attempts are allowed before login dies, but after the first three, the response starts to get very slow. Login failures are reported via the syslog facility. This facility is also used to report any successful root logins.

If the file .hushlogin exists, then a "quiet" login is performed (this disables the checking of mail and the printing of the last login time and message of the day). Otherwise, if /var/log/lastlog exists, the last login time is printed (and the current login is recorded).

Random administrative things, such as setting the UID and GID of the tty are performed. The TERM environment variable is preserved, if it exists (other environment variables are preserved if the -p option is used). Then the HOME, PATH, SHELL, TERM, MAIL, and LOGNAME environment variables are set. PATH defaults to /usr/local/bin:/bin:/usr/bin for normal users, and to /usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin for root. Last, if this is not a "quiet" login, the message of the day is printed and the file with the user's name in /var/spool/mail will be checked, and a message printed if it has non-zero length.

The user's shell is then started. If no shell is specified for the user in /etc/passwd, then /bin/sh is used. If there is no directory specified in /etc/passwd, then / is used (the home directory is checked for the .hushlogin file described above).

OPTIONS

-p Used by getty(8) to tell login not to destroy the environment

- f Used to skip a second login authentication. This specifically does not work for root, and does not appear to work well under Linux.
- h Used by other servers (i.e., telnetd(8)) to pass the name of the remote host to login so that it may be placed in utmp and wtmp. Only the superuser may use this option.

SPECIAL ACCESS RESTRICTIONS

The file /etc/securetty lists the names of the ttys where root is allowed to log in. One name of a tty device without the /dev/ prefix must be specified on each line. If the file does not exist, root is allowed to log in on any tty.

On most modern Linux systems PAM (Pluggable Authentication Modules) is used. On systems that do not use PAM, the file /etc/usertty specifies additional access restrictions for specific users. If this file does not exist, no additional access restrictions are imposed. The file consists of a sequence of sections. There are three possible section types: CLASSES, GROUPS and USERS. A CLASSES section defines classes of ttys and host-name patterns, A GROUPS section defines allowed ttys and hosts on a per group basis, and a USERS section defines allowed ttys and hosts on a per user basis.

Each line in this file in may be no longer than 255 characters. Comments start with # character and extend to the end of the line.

The CLASSES Section

A CLASSES section begins with the word CLASSES at the start of a line in all upper case. Each following line until the start of a new section or the end of the file consists of a sequence of words separated by tabs or spaces. Each line defines a class of ttys and host patterns.

The word at the beginning of a line becomes defined as a collective name for the ttys and host patterns specified at the rest of the line. This collective name can be used in any subsequent GROUPS or USERS section. No such class name must occur as part of the definition of a class in order to avoid problems with recursive classes.

An example CLASSES section:

```
CLASSES
myclass1      tty1 tty2
myclass2      tty3 @.foo.com
```

This defines the classes myclass1 and myclass2 as the corresponding right hand sides.

The GROUPS Section

A GROUPS section defines allowed ttys and hosts on a per Unix group basis. If a user is a member of a Unix group according to /etc/passwd and /etc/group and such a group is mentioned in a GROUPS section in /etc/usertty then the user is granted access if the group is.

A GROUPS section starts with the word GROUPS in all upper case at the start of a line, and each following line is a sequence of words separated by spaces or tabs. The first word on a line is the name of the group and the rest of the words on the line specifies the ttys and hosts where members of that group are allowed access. These specifications may involve the use of classes defined in previous CLASSES sections.

An example GROUPS section.

GROUPS

```
sys      tty1 @.bar.edu
stud     myclass1 tty4
```

This example specifies that members of group `sys` may log in on `tty1` and from hosts in the `bar.edu` domain. Users in group `stud` may log in from hosts/ttys specified in the class `myclass1` or from `tty4`.

The USERS Section

A `USERS` section starts with the word `USERS` in all upper case at the start of a line, and each following line is a sequence of words separated by spaces or tabs. The first word on a line is a username and that user is allowed to log in on the ttys and from the hosts mentioned on the rest of the line. These specifications may involve classes defined in previous `CLASSES` sections. If no section header is specified at the top of the file, the first section defaults to be a `USERS` section.

An example `USERS` section:

USERS

```
zacho      tty1 @130.225.16.0/255.255.255.0
blue       tty3 myclass2
```

This lets the user `zacho` login only on `tty1` and from hosts with IP addresses in the range `130.225.16.0 - 130.225.16.255`, and user `blue` is allowed to log in from `tty3` and whatever is specified in the class `myclass2`.

There may be a line in a `USERS` section starting with a username of `*`. This is a default rule and it will be applied to any user not matching any other line.

If both a USERS line and GROUPS line match a user then the user is allowed access from the union of all the ttys/hosts mentioned in these specifications.

Origins

The tty and host pattern specifications used in the specification of classes, group and user access are called origins. An origin string may have one of these formats:

- o The name of a tty device without the /dev/ prefix, for example tty1 or ttyS0.
- o The string @localhost, meaning that the user is allowed to telnet/rlogin from the local host to the same host. This also allows the user to for example run the command: xterm -e /bin/login.
- o A domain name suffix such as @.some.dom, meaning that the user may rlogin/telnet from any host whose domain name has the suffix .some.dom.
- o A range of IPv4 addresses, written @x.x.x.x/y.y.y.y where x.x.x.x is the IP address in the usual dotted quad decimal notation, and y.y.y.y is a bitmask in the same notation specifying which bits in the address to compare with the IP address of the remote host. For example @130.225.16.0/255.255.254.0 means that the user may rlogin/telnet from any host whose IP address is in the range 130.225.16.0 - 130.225.17.255.

Any of the above origins may be prefixed by a time specification according to the syntax:

```
timespec    ::= '[' <day-or-hour> [':' <day-or-hour>]* ']'
day         ::= 'mon' | 'tue' | 'wed' | 'thu' | 'fri' | 'sat' | 'sun'
hour        ::= '0' | '1' | ... | '23'
```

```
hourspec    ::= <hour> | <hour> '-' <hour>
day-or-hour ::= <day> | <hourspec>
```

For example, the origin [mon:tue:wed:thu:fri:8-17]tty3 means that log in is allowed on Mondays through Fridays between 8:00 and 17:59 (5:59 pm) on tty3. This also shows that an hour range a-b includes all moments between a:00 and b:59. A single hour specification (such as 10) means the time span between 10:00 and 10:59.

Not specifying any time prefix for a tty or host means log in from that origin is allowed any time. If you give a time prefix be sure to specify both a set of days and one or more hours or hour ranges. A time specification may not include any white space.

If no default rule is given then users not matching any line /etc/usertty are allowed to log in from anywhere as is standard behavior.

FILES

```
/var/run/utmp
/var/log/wtmp
/var/log/lastlog
/var/spool/mail/*
/etc/motd
/etc/passwd
/etc/nologin
/etc/usertty
.hushlogin
```

SEE ALSO

init(8), getty(8), mail(1), passwd(1), passwd(5), environ(7), shutdown(8)

BUGS

The undocumented BSD `-r` option is not supported. This may be required by some `rlogind(8)` programs.

A recursive login, as used to be possible in the good old days, no longer works; for most purposes `su(1)` is a satisfactory substitute. Indeed, for security reasons, login does a `vhangup()` system call to remove any possible listening processes on the tty. This is to avoid password sniffing. If one uses the command "login", then the surrounding shell gets killed by `vhangup()` because it's no longer the true owner of the tty. This can be avoided by using "exec login" in a top-level shell or xterm.

AUTHOR

Derived from BSD login 5.40 (5/9/89) by Michael Glad (glad@daimi.dk) for HP-UX
Ported to Linux 0.12: Peter Orbaek (poe@daimi.aau.dk)

login 分析

Login 程序主要可以分为以下几个主要部分：

1. Login 首先检查登录者是否为超级用户，如果不是超级用户，并且存在/etc/nologin 文件，则输出该文件内容，并中止登录过程；主要由 checknologin () 实现
2. 如果登录用户是超级用户，那么 login 必须在/etc/securetty/中指定的 tty 列表中实现登录，否则将导致登录失败。同样可以不指定/etc/securetty 文件，此时，超级用户可以在任何 tty 上登录
3. login 接下来将提示输入登录密码（由 getpass () 调用完成），并进行验证，如果密码不对，则提示重新登录
4. 顺利经过密码验证后，login 还将检查是否存在.hushlogin 文件，如果该文件存在，则执行一次"quiet"登录（所谓的 quiet 登录指的是，登录时不再提示邮件 mail，不再显示最后一次登录时间，不输出任何消息。启动级别为 3 时，正常情况下输出这些信息）
5. login 接下来设置登录 tty 的用户 ID 和组 ID，并设置相应的环境变量，包括 HOME、PATH、SHELL、TERM、LOGNAME 等。对于普通用户来说，PATH 缺省被设置成 /usr/local/bin:/bin:/usr/bin；对于超级用户来说，PATH 被设置成 /usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
6. login 的最后一步是为用户启动 shell。如果在/etc/passwd 中没有为用户指定 shell，那么将使用/bin/sh，如果在/etc/passwd 中没有给出当前工作目录，则使用"/"。

在 login 以后那就是你的天地了。

login 源码

```
1  /* This program is derived from 4.3 BSD software and is
2     subject to the copyright notice below.
3
4     The port to HP-UX has been motivated by the incapability
5     of 'rlogin'/'rlogind' as per HP-UX 6.5 (and 7.0) to transfer window sizes.
6
7     Changes:
8
9     - General HP-UX portation. Use of facilities not available
10       in HP-UX (e.g. setpriority) has been eliminated.
11       Utmp/wtmp handling has been ported.
12
13     - The program uses BSD command line options to be used
14       in connection with e.g. 'rlogind' i.e. 'new login'.
15
16     - HP features left out:      password expiry
17                               '*' as login shell, add it if you need it
18
19     - BSD features left out:      quota checks
20                                   password expiry
21                                   analysis of terminal type (tset feature)
22
```

```
23      - BSD features thrown in:      Security logging to syslogd.
24                                     This requires you to have a (ported) syslog
25                                     system -- 7.0 comes with syslog
26
27                                     'Lastlog' feature.
28
29      - A lot of nitty gritty details have been adjusted in favour of
30        HP-UX, e.g. /etc/securetty, default paths and the environment
31        variables assigned by 'login'.
32
33      - We do *nothing* to setup/alter tty state, under HP-UX this is
34        to be done by getty/rlogind/telnetd/some one else.
35
36      Michael Glad (glad@daimi.dk)
37      Computer Science Department
38      Aarhus University
39      Denmark
40
41      1990-07-04
42
43      1991-09-24 glad@daimi.aau.dk: HP-UX 8.0 port:
44      - now explicitly sets non-blocking mode on descriptors
45      - strcasecmp is now part of HP-UX
46
47      1992-02-05 poe@daimi.aau.dk: Ported the stuff to Linux 0.12
48      From 1992 till now (1997) this code for Linux has been maintained at
```

```
49      ftp.daimi.aau.dk:/pub/linux/poe/
50
51      1999-02-22 Arkadiusz Miśkiewicz <misiek@pld.ORG.PL>
52      - added Native Language Support
53      Sun Mar 21 1999 - Arnaldo Carvalho de Melo <acme@conectiva.com.br>
54      - fixed strerr(errno) in gettext calls
55  */
56
57 /*
58  * Copyright (c) 1980, 1987, 1988 The Regents of the University of California.
59  * All rights reserved.
60  *
61  * Redistribution and use in source and binary forms are permitted
62  * provided that the above copyright notice and this paragraph are
63  * duplicated in all such forms and that any documentation,
64  * advertising materials, and other materials related to such
65  * distribution and use acknowledge that the software was developed
66  * by the University of California, Berkeley. The name of the
67  * University may not be used to endorse or promote products derived
68  * from this software without specific prior written permission.
69  * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
70  * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
71  * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
72  */
73
74 /*
```



```
75  * login [ name ]
76  * login -h hostname (for telnetd, etc.)
77  * login -f name (for pre-authenticated login: datakit, xterm, etc.)
78  */
79
80  /* #define TESTING */
81
82  #ifdef TESTING
83  #include "param.h"
84  #else
85  #include <sys/param.h>
86  #endif
87
88  #include <stdio.h>
89  #include <ctype.h>
90  #include <unistd.h>
91  #include <getopt.h>
92  #include <memory.h>
93  #include <time.h>
94  #include <sys/stat.h>
95  #include <sys/time.h>
96  #include <sys/resource.h>
97  #include <sys/file.h>
98  #include <termios.h>
99  #include <string.h>
100 #define index strchr
```

```
101  #define rindex strchr
102  #include <sys/ioctl.h>
103  #include <sys/wait.h>
104  #include <signal.h>
105  #include <errno.h>
106  #include <grp.h>
107  #include <pwd.h>
108  #include <utmp.h>
109  #include <setjmp.h>
110  #include <stdlib.h>
111  #include <string.h>
112  #include <sys/syslog.h>
113  #include <sys/sysmacros.h>
114  #include <netdb.h>
115  #include "pathnames.h"
116  #include "my_crypt.h"
117  #include "login.h"
118  #include "xstrncpy.h"
119  #include "nls.h"
120
121  #ifdef __linux__
122  #   include <sys/sysmacros.h>
123  #   include <linux/major.h>
124  #endif
125
126  #ifdef TESTING
```

```
127  # include "utmp.h"
128  #else
129  # include <utmp.h>
130  #endif
131
132  #ifdef SHADOW_PWD
133  # include <shadow.h>
134  #endif
135
136  #ifdef USE_PAM
137  # include <security/pam_appl.h>
138  # include <security/pam_misc.h>
139  # define PAM_MAX_LOGIN_TRIES    3
140  # define PAM_FAIL_CHECK if (retcode != PAM_SUCCESS) { \
141      fprintf(stderr, "\n%s\n", pam_strerror(pamh, retcode)); \
142      syslog(LOG_ERR, "%s", pam_strerror(pamh, retcode)); \
143      pam_end(pamh, retcode); exit(1); \
144  }
145  # define PAM_END { \
146      pam_setcred(pamh, PAM_DELETE_CRED); \
147      retcode = pam_close_session(pamh, 0); \
148      pam_end(pamh, retcode); \
149  }
150  #endif
151
152  #ifndef __linux__
```

```
153  # include <tzfile.h>
154  #endif
155  #include <lastlog.h>
156
157  #define SLEEP_EXIT_TIMEOUT 5
158
159  #ifdef __linux__
160  #define DO_PS_FIDDLING
161  #endif
162
163  #ifdef DO_PS_FIDDLING
164  #include "setproctitle.h"
165  #endif
166
167  #if 0
168  /* from before we had a lastlog.h file in linux */
169  struct lastlog
170  { long ll_time;
171    char ll_line[12];
172    char ll_host[16];
173  };
174  #endif
175
176  #ifndef USE_PAM
177  static void getloginname (void);
178  static void checknologin (void);
```

```
179 static int rootterm (char *ttyn);
180 #endif
181 static void timedout (int);
182 static void sigint (int);
183 static void motd (void);
184 static void dolastlog (int quiet);
185
186 #ifdef CRYPTOCARD
187 #include "cryptocard.h"
188 #endif
189
190 #ifdef KERBEROS
191 #include <kerberos/krb.h>
192 #include <sys/termios.h>
193 char realm[REALM_SZ];
194 int kerror = KSUCCESS, notickets = 1;
195 #endif
196
197 #ifdef USE_TTY_GROUP
198 # define TTY_MODE 0620
199 #else
200 # define TTY_MODE 0600
201 #endif
202
203 #define TTYGRPNAME "tty" /* name of group to own ttys */
204
```

```
205  #ifndef MAXPATHLEN
206  #   define MAXPATHLEN 1024
207  #endif
208
209  /*
210   * This bounds the time given to login. Not a define so it can
211   * be patched on machines where it's too small.
212   */
213  #ifndef __linux__
214  int timeout = 300;
215  #else
216  int    timeout = 60;    /* used in cryptocard.c */
217  #endif
218
219  struct passwd *pwd;      /* used in cryptocard.c */
220  #if USE_PAM
221  static struct passwd pwdcopy;
222  #endif
223  char    hostaddress[4];    /* used in checktty.c */
224  char    *hostname;        /* idem */
225  static char    *username, *tty_name, *tty_number;
226  static char    thishost[100];
227  static int failures = 1;
228  static pid_t  pid;
229
230  #ifndef __linux__
```

```
231 struct sgttyb sgttyb;
232 struct tchars tc = {
233     CINTR, CQUIT, CSTART, CSTOP, CEOT, CBRK
234 };
235 struct ltchars ltc = {
236     CSUSP, CDSUSP, CRPRNT, CFLUSH, CWERASE, CLNEXT
237 };
238 #endif
239
240 /* Nice and simple code provided by Linus Torvalds 16-Feb-93 */
241 /* Nonblocking stuff by Maciej W. Rozycki, macro@ds2.pg.gda.pl, 1999.
242    He writes: "Login performs open() on a tty in a blocking mode.
243    In some cases it may make login wait in open() for carrier infinitely,
244    for example if the line is a simplistic case of a three-wire serial
245    connection. I believe login should open the line in the non-blocking mode
246    leaving the decision to make a connection to getty (where it actually
247    belongs). */
248 static void
249 opentty(const char * tty) {
250     int i, fd, flags;
251
252     fd = open(tty, O_RDWR | O_NONBLOCK);
253     if (fd == -1) {
254         syslog(LOG_ERR, _("FATAL: can't reopen tty: %s"),
255             strerror(errno));
256         sleep(1);
```

```
257         exit(1);
258     }
259
260     flags = fcntl(fd, F_GETFL);
261     flags &= ~O_NONBLOCK;
262     fcntl(fd, F_SETFL, flags);
263
264     for (i = 0; i < fd; i++)
265         close(i);
266     for (i = 0; i < 3; i++)
267         if (fd != i)
268             dup2(fd, i);
269     if (fd >= 3)
270         close(fd);
271 }
272
273 /* In case login is suid it was possible to use a hardlink as stdin
274    and exploit races for a local root exploit. (Wojciech Purczynski). */
275 /* More precisely, the problem is ttyn := ttyname(0); ...; chown(ttyn);
276    here ttyname() might return "/tmp/x", a hardlink to a pseudotty. */
277 /* All of this is a problem only when login is suid, which it isnt. */
278 static void
279 check_ttyname(char *ttyn) {
280     struct stat statbuf;
281
282     if (lstat(ttyn, &statbuf)
```



```
283         || !S_ISCHR(statbuf.st_mode)
284         || (statbuf.st_nlink > 1 && strncmp(ttyn, "/dev/", 5))) {
285     syslog(LOG_ERR, _("FATAL: bad tty"));
286     sleep(1);
287     exit(1);
288 }
289 }
290
291 /* true if the filedescriptor fd is a console tty, very Linux specific */
292 static int
293 consoletty(int fd) {
294     #ifdef __linux__
295         struct stat stb;
296
297         if ((fstat(fd, &stb) >= 0)
298             && (major(stb.st_rdev) == TTY_MAJOR)
299             && (minor(stb.st_rdev) < 64)) {
300             return 1;
301         }
302     #endif
303     return 0;
304 }
305
306 #if USE_PAM
307 /*
308  * Log failed login attempts in _PATH_BTMP if that exists.
```

```
309  * Must be called only with username the name of an actual user.
310  * The most common login failure is to give password instead of username.
311  */
312  #define    _PATH_BTMP "/var/log/btmp"
313  static void
314  logbtmp(const char *line, const char *username, const char *hostname) {
315      struct utmp ut;
316
317      memset(&ut, 0, sizeof(ut));
318
319      strncpy(ut.ut_user, username ? username : "(unknown)",
320              sizeof(ut.ut_user));
321
322      strncpy(ut.ut_id, line + 3, sizeof(ut.ut_id));
323      xstrncpy(ut.ut_line, line, sizeof(ut.ut_line));
324
325      #if defined(_HAVE_UT_TV)    /* in <utmpbits.h> included by <utmp.h> */
326          gettimeofday(&ut.ut_tv, NULL);
327      #else
328          {
329              time_t t;
330              time(&t);
331              ut.ut_time = t;    /* ut_time is not always a time_t */
332          }
333      #endif
334
```

```
335     ut.ut_type = LOGIN_PROCESS; /* XXX doesn't matter */
336     ut.ut_pid = pid;
337     if (hostname) {
338         xstrncpy(ut.ut_host, hostname, sizeof(ut.ut_host));
339         if (hostaddress[0])
340             memcpy(&ut.ut_addr, hostaddress, sizeof(ut.ut_addr));
341     }
342     #ifdef HAVE_updwtmp      /* bad luck for ancient systems */
343         updwtmp(_PATH_BTMP, &ut);
344     #endif
345 }
346 #endif /* USE_PAM */
347
348 int
349 main(int argc, char **argv)
350 {
351     extern int optind;
352     extern char *optarg, **environ;
353     struct group *gr;
354     register int ch;
355     register char *p;
356     int ask, fflag, hflag, pflag, cnt, errsv;
357     int quietlog, passwd_req;
358     char *domain, *ttyn;
359     char tbuf[MAXPATHLEN + 2], tname[sizeof(_PATH_TTY) + 10];
360     char *termenv;
```

```
361     char *childArgv[10];
362     char *buff;
363     int childArgc = 0;
364     #ifdef USE_PAM
365         int retcode;
366         pam_handle_t *pamh = NULL;
367         struct pam_conv conv = { misc_conv, NULL };
368         pid_t childPid;
369     #else
370         char *salt, *pp;
371     #endif
372     #ifdef CHOWNVCS
373         char vcsn[20], vcsan[20];
374     #endif
375
376     pid = getpid();
377
378     signal(SIGALRM, timedout);
379     alarm((unsigned int)timeout);
380     signal(SIGQUIT, SIG_IGN);
381     signal(SIGINT, SIG_IGN);
382
383     setlocale(LC_ALL, "");
384     bindtextdomain(PACKAGE, LOCALEDIR);
385     textdomain(PACKAGE);
386
```

```
387     setpriority(PRIO_PROCESS, 0, 0);
388 #ifdef HAVE_QUOTA
389     quota(Q_SETTID, 0, 0, 0);
390 #endif
391 #ifdef DO_PS_FIDDLING
392     initproctitle(argc, argv);
393 #endif
394
395     /*
396      * -p is used by getty to tell login not to destroy the environment
397      * -f is used to skip a second login authentication
398      * -h is used by other servers to pass the name of the remote
399      *   host to login so that it may be placed in utmp and wtmp
400      */
401     gethostname(tbuf, sizeof(tbuf));
402     xstrncpy(thishost, tbuf, sizeof(thishost));
403     domain = index(tbuf, '.');
404
405     username = tty_name = hostname = NULL;
406     fflag = hflag = pflag = 0;
407     passwd_req = 1;
408
409     while ((ch = getopt(argc, argv, "fh:p")) != -1)
410         switch (ch) {
411             case 'f':
412                 fflag = 1;
413             case 'p':
414                 pflag = 1;
415             case 'h':
416                 hflag = 1;
417         }
418
419     /* login 接收三个选项：
420      * -p  Used by getty(8) to tell login not to destroy the environment
421      * -f  Used to skip a second login authentication. This specifically does not work
422      * -h  Used to pass the name of the remote host to login so that it may be placed in utmp and wtmp
423      */
```

```

413         break;                                for root, and does not appear to work well under Linux.
414                                     -h    Used by other servers (i.e., telnetd(8)) to pass the name of the remote host to
415     case 'h':                                login so that it may be placed in utmp and wtmp. Only the superuser may use this
416         if (getuid()) {                        option.
417             fprintf(stderr,
418                 _("login: -h for super-user only.\n"));
419             exit(1);
420         }
421         hflag = 1;
422         if (domain && (p = index(optarg, '.')) &&
423             strcasecmp(p, domain) == 0)
424             *p = 0;
425
426         hostname = strdup(optarg);    /* strdup: Ambrose C. Li */
427         {
428             struct hostent *he = gethostbyname(hostname);
429
430             /* he points to static storage; copy the part we use */
431             hostaddress[0] = 0;
432             if (he && he->h_addr_list && he->h_addr_list[0])
433                 memcpy(hostaddress, he->h_addr_list[0],
434                     sizeof(hostaddress));
435         }
436         break;
437
438     case 'p':

```

```

439     pflag = 1;
440     break;
441
442     case '?':
443     default:
444         fprintf(stderr,
445             _("usage: login [-fp] [username]\n"));
446         exit(1);
447     }
448     argc -= optind;
449     argv += optind;
450     if (*argv) {                                检查 login 时是否提供了用户名，比如是 login 还是 login wzhou
451         char *p = *argv;
452         username = strdup(p);                    获得登录的用户名
453         ask = 0;
454         /* wipe name - some people mistype their password here */
455         /* (of course we are too late, but perhaps this helps a little ..) */
456         while(*p)
457             *p++ = ' ';
458     } else
459         ask = 1;                                运行 login 时没有提供用户名，则置需要向用户提问要用户名的标志
460
461     for (cnt = getdtablesize(); cnt > 2; cnt--)
462         close(cnt);
463
464     ttyn = ttyname(0);                          返回 login process file handle 0(标准输入)对应的 tty 设备名

```

```
465
466     if (ttyn == NULL || *ttyn == '\\0') {
467         /* no snprintf required - see definition of tname */
468         sprintf(tname, "%s??", _PATH_TTY);
469         ttyn = tname;
470     }
471
472     check_ttyname(ttyn);
473
474     if (strncmp(ttyn, "/dev/", 5) == 0)
475         tty_name = ttyn+5;
476     else
477         tty_name = ttyn;
478
479     if (strncmp(ttyn, "/dev/tty", 8) == 0)
480         tty_number = ttyn+8;
481     else {
482         char *p = ttyn;
483         while (*p && !isdigit(*p)) p++;
484         tty_number = p;
485     }
486
487 #ifndef CHOWNVCS
488     /* find names of Virtual Console devices, for later mode change */
489     snprintf(vcsn, sizeof(vcsn), "/dev/vcs%s", tty_number);
490     snprintf(vcsan, sizeof(vcsan), "/dev/vcsa%s", tty_number);
```



```
491  #endif
492
493      /* set pgid to pid */
494      setpgrp();
495      /* this means that setsid() will fail */
496
497      {
498          struct termios tt, ttt;
499
500          tcgetattr(0, &tt);
501          ttt = tt;
502          ttt.c_cflag &= ~HUPCL;
503
504          /* These can fail, e.g. with ttyn on a read-only filesystem */
505          chown(ttyn, 0, 0);
506          chmod(ttyn, TTY_MODE);
507
508          /* Kill processes left on this tty */
509          tcsetattr(0, TCSAFLUSH, &ttt);
510          signal(SIGHUP, SIG_IGN); /* so vhangup() wont kill us */      忽略 SIGHUP signal
511          vhangup();
512          signal(SIGHUP, SIG_DFL);
513
514          /* open stdin, stdout, stderr to the tty */
515          opentty(ttyn);          该函数使得 file handle 的 0, 1, 2 都指向 tty
516
```

```
517     /* restore tty modes */
518     tcsetattr(0,TCSAFLUSH,&tt);
519 }
520
521     openlog("login", LOG_ODELAY, LOG_AUTHPRIV);
522
523     #if 0
524         /* other than iso-8859-1 */
525         printf("\033(K");
526         fprintf(stderr, "\033(K");
527     #endif
528
529     #ifdef USE_PAM
530         /*
531          * username is initialized to NULL
532          * and if specified on the command line it is set.
533          * Therefore, we are safe not setting it to anything
534          */
535
536         retcode = pam_start("login",username, &conv, &pamh);
537         if(retcode != PAM_SUCCESS) {
538             fprintf(stderr, _("login: PAM Failure, aborting: %s\n"),
539                 pam_strerror(pamh, retcode));
540             syslog(LOG_ERR, _("Couldn't initialize PAM: %s"),
541                 pam_strerror(pamh, retcode));
542             exit(99);
543         }
544     #endif
```

L529 到 L692 都是 PAM 相关的认证，关于 PAM 另文介绍，这是 Linux 凌乱的安全体系中的一种很灵活的安全架构。本文只分析经典（也是原始）的认证

```
543     }
544     /* hostname & tty are either set to NULL or their correct values,
545        depending on how much we know */
546     retcode = pam_set_item(pamh, PAM_RHOST, hostname);
547     PAM_FAIL_CHECK;
548     retcode = pam_set_item(pamh, PAM_TTY, tty_name);
549     PAM_FAIL_CHECK;
550
551     /*
552      * Andrew.Taylor@cal.montage.ca: Provide a user prompt to PAM
553      * so that the "login: " prompt gets localized. Unfortunately,
554      * PAM doesn't have an interface to specify the "Password: " string
555      * (yet).
556      */
557     retcode = pam_set_item(pamh, PAM_USER_PROMPT, _("login: "));
558     PAM_FAIL_CHECK;
559
560     #if 0
561     /*
562      * other than iso-8859-1
563      * one more time due to reset tty by PAM
564      */
565     printf("\033(K");
566     fprintf(stderr, "\033(K");
567     #endif
568
```

```
569      /* if fflag == 1, then the user has already been authenticated */
570      if (fflag && (getuid() == 0))
571          passwd_req = 0;
572      else
573          passwd_req = 1;
574
575      if(passwd_req == 1) {
576          int failcount=0;
577
578          /* if we didn't get a user on the command line, set it to NULL */
579          pam_get_item(pamh, PAM_USER, (const void **) &username);
580          if (!username)
581              pam_set_item(pamh, PAM_USER, NULL);
582
583          /* there may be better ways to deal with some of these
584             conditions, but at least this way I don't think we'll
585             be giving away information... */
586          /* Perhaps someday we can trust that all PAM modules will
587             pay attention to failure count and get rid of MAX_LOGIN_TRIES? */
588
589          retcode = pam_authenticate(pamh, 0);
590          while((failcount++ < PAM_MAX_LOGIN_TRIES) &&
591              ((retcode == PAM_AUTH_ERR) ||
592              (retcode == PAM_USER_UNKNOWN) ||
593              (retcode == PAM_CRED_INSUFFICIENT) ||
594              (retcode == PAM_AUTHINFO_UNAVAIL))) {
```

```
595     pam_get_item(pamh, PAM_USER, (const void **) &username);
596
597     syslog(LOG_NOTICE,_( "FAILED LOGIN %d FROM %s FOR %s, %s"),
598         failcount, hostname, username, pam_strerror(pamh, retcode));
599     logbtmp(tty_name, username, hostname);
600
601     fprintf(stderr,_( "Login incorrect\n\n"));
602     pam_set_item(pamh,PAM_USER,NULL);
603     retcode = pam_authenticate(pamh, 0);
604 }
605
606 if (retcode != PAM_SUCCESS) {
607     pam_get_item(pamh, PAM_USER, (const void **) &username);
608
609     if (retcode == PAM_MAXTRIES)
610         syslog(LOG_NOTICE,_( "TOO MANY LOGIN TRIES (%d) FROM %s FOR "
611             "%s, %s"), failcount, hostname, username,
612             pam_strerror(pamh, retcode));
613     else
614         syslog(LOG_NOTICE,_( "FAILED LOGIN SESSION FROM %s FOR %s, %s"),
615             hostname, username, pam_strerror(pamh, retcode));
616     logbtmp(tty_name, username, hostname);
617
618     fprintf(stderr,_( "\nLogin incorrect\n"));
619     pam_end(pamh, retcode);
620     exit(0);
```

```
621     }
622
623     retcode = pam_acct_mgmt(pamh, 0);
624
625     if(retcode == PAM_NEW_AUTHTOK_REQD) {
626         retcode = pam_chauthtok(pamh, PAM_CHANGE_EXPIRED_AUTHTOK);
627     }
628
629     PAM_FAIL_CHECK;
630 }
631
632 /*
633  * Grab the user information out of the password file for future usage
634  * First get the username that we are actually using, though.
635  */
636 retcode = pam_get_item(pamh, PAM_USER, (const void **) &username);
637 PAM_FAIL_CHECK;
638
639 if (!username || !*username) {
640     fprintf(stderr, _("\nSession setup problem, abort.\n"));
641     syslog(LOG_ERR, _("NULL user name in %s:%d. Abort."),
642         __FUNCTION__, __LINE__);
643     pam_end(pamh, PAM_SYSTEM_ERR);
644     exit(1);
645 }
646 if (!(pwd = getpwnam(username))) {
```

```
647         fprintf(stderr, _("\nSession setup problem, abort.\n"));
648         syslog(LOG_ERR, _("Invalid user name \"%s\" in %s:%d. Abort."),
649             username, __FUNCTION__, __LINE__);
650         pam_end(pamh, PAM_SYSTEM_ERR);
651         exit(1);
652     }
653
654     /*
655     * Create a copy of the pwd struct - otherwise it may get
656     * clobbered by PAM
657     */
658     memcpy(&pwdcopy, pwd, sizeof(*pwd));
659     pwd = &pwdcopy;
660     pwd->pw_name = strdup(pwd->pw_name);
661     pwd->pw_passwd = strdup(pwd->pw_passwd);
662     pwd->pw_gecos = strdup(pwd->pw_gecos);
663     pwd->pw_dir = strdup(pwd->pw_dir);
664     pwd->pw_shell = strdup(pwd->pw_shell);
665     if (!pwd->pw_name || !pwd->pw_passwd || !pwd->pw_gecos ||
666         !pwd->pw_dir || !pwd->pw_shell) {
667         fprintf(stderr, _("login: Out of memory\n"));
668         syslog(LOG_ERR, "Out of memory");
669         pam_end(pamh, PAM_SYSTEM_ERR);
670         exit(1);
671     }
672     username = pwd->pw_name;
```

```
673
674     /*
675     * Initialize the supplementary group list.
676     * This should be done before pam_setcred because
677     * the PAM modules might add groups during pam_setcred.
678     */
679     if (initgroups(username, pwd->pw_gid) < 0) {
680         syslog(LOG_ERR, "initgroups: %m");
681         fprintf(stderr, _("\nSession setup problem, abort.\n"));
682         pam_end(pamh, PAM_SYSTEM_ERR);
683         exit(1);
684     }
685
686     retcode = pam_open_session(pamh, 0);
687     PAM_FAIL_CHECK;
688
689     retcode = pam_setcred(pamh, PAM_ESTABLISH_CRED);
690     PAM_FAIL_CHECK;
691
692     #else /* ! USE_PAM */          非 PAM 认证
693
694     for (cnt = 0;; ask = 1) {
695
696         if (ask) {
697             fflag = 0;
698             getloginname();          在终端上输出“login: ”,获得用户登录名,由全局变量 username
```



```

699     }
700
701     /* Dirty patch to fix a gigantic security hole when using
702        yellow pages. This problem should be solved by the
703        libraries, and not by programs, but this must be fixed
704        urgently! If the first char of the username is '+', we
705        avoid login success.
706        Feb 95 <alvaro@etsit.upm.es> */
707
708     if (username[0] == '+') {
709         puts(_("Illegal username"));
710         badlogin(username);
711         sleepexit(1);
712     }
713
714     /* (void)strcpy(tbuf, username); why was this here? */
715     if ((pwd = getpwnam(username))) {           从/etc/passwd 中获得与登录用户相关信息，存放在 pwd ( struct passwd 结构中
716
717         struct passwd {
718             char    *pw_name;        /* user name */
719             char    *pw_passwd;      /* user password */
720             uid_t   pw_uid;          /* user id */
721             gid_t   pw_gid;          /* group id */
722             char    *pw_gecos;       /* real name */
723             char    *pw_dir;         /* home directory */
724             char    *pw_shell;       /* shell program */
725         };
726     }
727
728     if (pwd == NULL)
729         return 0;
730
731     /* If the user is not root, we must check if the user is
732        allowed to login. This is done by checking if the user
733        is in the 'passwd' file. If the user is not in the
734        'passwd' file, we return 0. If the user is in the
735        'passwd' file, we return 1. */
736     if (strcmp(pwd->pw_name, username) != 0)
737         return 0;
738
739     /* If the user is root, we return 1. */
740     if (strcmp(pwd->pw_name, "root") == 0)
741         return 1;
742
743     /* If the user is not root, we return 0. */
744     return 0;
745 }
746
747 int main(int argc, char **argv)
748 {
749     int i;
750     int n;
751     int r;
752     int s;
753     int t;
754     int u;
755     int v;
756     int w;
757     int x;
758     int y;
759     int z;
760     int a;
761     int b;
762     int c;
763     int d;
764     int e;
765     int f;
766     int g;
767     int h;
768     int i;
769     int j;
770     int k;
771     int l;
772     int m;
773     int n;
774     int o;
775     int p;
776     int q;
777     int r;
778     int s;
779     int t;
780     int u;
781     int v;
782     int w;
783     int x;
784     int y;
785     int z;
786     int a;
787     int b;
788     int c;
789     int d;
790     int e;
791     int f;
792     int g;
793     int h;
794     int i;
795     int j;
796     int k;
797     int l;
798     int m;
799     int n;
800     int o;
801     int p;
802     int q;
803     int r;
804     int s;
805     int t;
806     int u;
807     int v;
808     int w;
809     int x;
810     int y;
811     int z;
812     int a;
813     int b;
814     int c;
815     int d;
816     int e;
817     int f;
818     int g;
819     int h;
820     int i;
821     int j;
822     int k;
823     int l;
824     int m;
825     int n;
826     int o;
827     int p;
828     int q;
829     int r;
830     int s;
831     int t;
832     int u;
833     int v;
834     int w;
835     int x;
836     int y;
837     int z;
838     int a;
839     int b;
840     int c;
841     int d;
842     int e;
843     int f;
844     int g;
845     int h;
846     int i;
847     int j;
848     int k;
849     int l;
850     int m;
851     int n;
852     int o;
853     int p;
854     int q;
855     int r;
856     int s;
857     int t;
858     int u;
859     int v;
860     int w;
861     int x;
862     int y;
863     int z;
864     int a;
865     int b;
866     int c;
867     int d;
868     int e;
869     int f;
870     int g;
871     int h;
872     int i;
873     int j;
874     int k;
875     int l;
876     int m;
877     int n;
878     int o;
879     int p;
880     int q;
881     int r;
882     int s;
883     int t;
884     int u;
885     int v;
886     int w;
887     int x;
888     int y;
889     int z;
890     int a;
891     int b;
892     int c;
893     int d;
894     int e;
895     int f;
896     int g;
897     int h;
898     int i;
899     int j;
900     int k;
901     int l;
902     int m;
903     int n;
904     int o;
905     int p;
906     int q;
907     int r;
908     int s;
909     int t;
910     int u;
911     int v;
912     int w;
913     int x;
914     int y;
915     int z;
916     int a;
917     int b;
918     int c;
919     int d;
920     int e;
921     int f;
922     int g;
923     int h;
924     int i;
925     int j;
926     int k;
927     int l;
928     int m;
929     int n;
930     int o;
931     int p;
932     int q;
933     int r;
934     int s;
935     int t;
936     int u;
937     int v;
938     int w;
939     int x;
940     int y;
941     int z;
942     int a;
943     int b;
944     int c;
945     int d;
946     int e;
947     int f;
948     int g;
949     int h;
950     int i;
951     int j;
952     int k;
953     int l;
954     int m;
955     int n;
956     int o;
957     int p;
958     int q;
959     int r;
960     int s;
961     int t;
962     int u;
963     int v;
964     int w;
965     int x;
966     int y;
967     int z;
968     int a;
969     int b;
970     int c;
971     int d;
972     int e;
973     int f;
974     int g;
975     int h;
976     int i;
977     int j;
978     int k;
979     int l;
980     int m;
981     int n;
982     int o;
983     int p;
984     int q;
985     int r;
986     int s;
987     int t;
988     int u;
989     int v;
990     int w;
991     int x;
992     int y;
993     int z;
994     int a;
995     int b;
996     int c;
997     int d;
998     int e;
999     int f;
1000    int g;
1001    int h;
1002    int i;
1003    int j;
1004    int k;
1005    int l;
1006    int m;
1007    int n;
1008    int o;
1009    int p;
1010    int q;
1011    int r;
1012    int s;
1013    int t;
1014    int u;
1015    int v;
1016    int w;
1017    int x;
1018    int y;
1019    int z;
1020    int a;
1021    int b;
1022    int c;
1023    int d;
1024    int e;
1025    int f;
1026    int g;
1027    int h;
1028    int i;
1029    int j;
1030    int k;
1031    int l;
1032    int m;
1033    int n;
1034    int o;
1035    int p;
1036    int q;
1037    int r;
1038    int s;
1039    int t;
1040    int u;
1041    int v;
1042    int w;
1043    int x;
1044    int y;
1045    int z;
1046    int a;
1047    int b;
1048    int c;
1049    int d;
1050    int e;
1051    int f;
1052    int g;
1053    int h;
1054    int i;
1055    int j;
1056    int k;
1057    int l;
1058    int m;
1059    int n;
1060    int o;
1061    int p;
1062    int q;
1063    int r;
1064    int s;
1065    int t;
1066    int u;
1067    int v;
1068    int w;
1069    int x;
1070    int y;
1071    int z;
1072    int a;
1073    int b;
1074    int c;
1075    int d;
1076    int e;
1077    int f;
1078    int g;
1079    int h;
1080    int i;
1081    int j;
1082    int k;
1083    int l;
1084    int m;
1085    int n;
1086    int o;
1087    int p;
1088    int q;
1089    int r;
1090    int s;
1091    int t;
1092    int u;
1093    int v;
1094    int w;
1095    int x;
1096    int y;
1097    int z;
1098    int a;
1099    int b;
1100    int c;
1101    int d;
1102    int e;
1103    int f;
1104    int g;
1105    int h;
1106    int i;
1107    int j;
1108    int k;
1109    int l;
1110    int m;
1111    int n;
1112    int o;
1113    int p;
1114    int q;
1115    int r;
1116    int s;
1117    int t;
1118    int u;
1119    int v;
1120    int w;
1121    int x;
1122    int y;
1123    int z;
1124    int a;
1125    int b;
1126    int c;
1127    int d;
1128    int e;
1129    int f;
1130    int g;
1131    int h;
1132    int i;
1133    int j;
1134    int k;
1135    int l;
1136    int m;
1137    int n;
1138    int o;
1139    int p;
1140    int q;
1141    int r;
1142    int s;
1143    int t;
1144    int u;
1145    int v;
1146    int w;
1147    int x;
1148    int y;
1149    int z;
1150    int a;
1151    int b;
1152    int c;
1153    int d;
1154    int e;
1155    int f;
1156    int g;
1157    int h;
1158    int i;
1159    int j;
1160    int k;
1161    int l;
1162    int m
```

```

};

)
716 # ifdef SHADOW_PWD                                从/etc/shadow 中获得与登录用户相关信息
717     struct spwd *sp;
718
719     if ((sp = getspnam(username)))

struct spwd
{
    char *sp_namp;                /* login name */
    char *sp_pwdp;                /* encrypted password */
    sptime sp_lstchg;             /* date of last change */
    sptime sp_min;                /* minimum number of days between changes */
    sptime sp_max;                /* maximum number of days between changes */
    sptime sp_warn;               /* number of days of warning before password
                                expires */
    sptime sp_inact;              /* number of days after password expires
                                until the account becomes unusable. */
    sptime sp_expire;             /* days since 1/1/70 until account expires*/
    unsigned long sp_flag;        /* reserved for future use */
};

```

```
720         pwd->pw_passwd = sp->sp_pwdp;
721     # endif
722         salt = pwd->pw_passwd;
723     } else
724         salt = "xx";
725
726     if (pwd) {
727         initgroups(username, pwd->pw_gid);
728         checktty(username, tty_name, pwd); /* in checktty.c */
729     }
730
731     /* if user not super-user, check for disabled logins */
732     if (pwd == NULL || pwd->pw_uid)
733         checknologin();    检查有无/etc/nologin 文件，如果有则表示禁止该用户登录，输出/etc/nologin 中的内容
734
735     /*
736     * Disallow automatic login to root; if not invoked by
737     * root, disallow if the uid's differ.
738     */
739     if (fflag && pwd) {
740         int uid = getuid();
741
742         passwd_req = pwd->pw_uid == 0 ||
743             (uid && uid != pwd->pw_uid);
744     }
745
```

```

746      /*
747      * If trying to log in as root, but with insecure terminal,
748      * refuse the login attempt.
749      */
750      if (pwd && pwd->pw_uid == 0 && !rootterm(tty_name)) {
751          fprintf(stderr,
752              _("%s login refused on this terminal.\n"),
753              pwd->pw_name);
754
755          if (hostname)
756              syslog(LOG_NOTICE,
757                  _("LOGIN %s REFUSED FROM %s ON TTY %s"),
758                  pwd->pw_name, hostname, tty_name);
759          else
760              syslog(LOG_NOTICE,
761                  _("LOGIN %s REFUSED ON TTY %s"),
762                  pwd->pw_name, tty_name);
763          continue;
764      }
765
766      /*
767      * If no pre-authentication and a password exists
768      * for this user, prompt for one and verify it.
769      */
770      if (!passwd_req || (pwd && !*pwd->pw_passwd))
771          break;

```

如果是 root 用户登录则检查是否是通过安全的 tty 登录 ,不是则拒绝登录。所谓安全的 tty 登录就是是否在文件/etc/securetty 中被记录了

如果密码为空,也可以,就像
root::0:0:root:/root:/bin/bash

```

772
773     setpriority(PRIO_PROCESS, 0, -4);
774     pp = getpass(_("Password: "));
775
776 # ifdef CRYPTOCARD
777     if (strncmp(pp, "CRYPTO", 6) == 0) {
778         if (pwd && cryptocard()) break;
779     }
780 # endif /* CRYPTOCARD */
781
782     p = crypt(pp, salt);
783     setpriority(PRIO_PROCESS, 0, 0);
784
785 # ifdef KERBEROS
786     /*
787      * If not present in pw file, act as we normally would.
788      * If we aren't Kerberos-authenticated, try the normal
789      * pw file for a password. If that's ok, log the user
790      * in without issueing any tickets.
791      */
792
793     if (pwd && !krb_get_lrealm(realm,1)) {
794         /*
795          * get TGT for local realm; be careful about uid's
796          * here for ticket file ownership
797          */

```

中的密码项为空

提升本 process 的优先级 (为了获得密码, 还特意提升进程优先级)

在终端上显示 “ Password: ”, 获得用户输入的密码, 由 pp 指向

对用户输入的 passwd 进行加密, 获得密文

恢复本 process 的优先级

Kerberos 认证

```

798         setreuid(geteuid(),pwd->pw_uid);
799         kerror = krb_get_pw_in_tkt(pwd->pw_name, "", realm,
800             "krbtgt", realm, DEFAULT_TKT_LIFE, pp);
801         setuid(0);
802         if (kerror == INTK_OK) {
803             memset(pp, 0, strlen(pp));
804             notickets = 0;    /* user got ticket */
805             break;
806         }
807     }
808 # endif /* KERBEROS */
809     memset(pp, 0, strlen(pp));
810
811     if (pwd && !strcmp(p, pwd->pw_passwd))           比较两个密文是否相同
812         break;                                       相同则 break 出 L694 行的循环，即认证通过，到 L829 行运行
813
814     printf(_("Login incorrect\n"));                  输出 "Login incorrect"，表示密码不对
815     badlogin(username); /* log ALL bad logins */    在 syslog 中记录登录失败记录
816     failures++;
817
818     /* we allow 10 tries, but after 3 we start backing off */
819     if (++cnt > 3) {
820         if (cnt >= 10) {
821             sleepexit(1);                            超过 10 次登录都失败，则退出 login
822         }
823         sleep((unsigned int)((cnt - 3) * 5));

```

```

824     }
825     }
826     #endif /* !USE_PAM */
827     登录用户合法
828     /* committed to login -- turn off timeout */
829     alarm((unsigned int)0);          关闭所有闹钟
830
831     #ifdef HAVE_QUOTA                如果系统对登录用户数有限制
832     if (quota(Q_SETUID, pwd->pw_uid, 0, 0) < 0 && errno != EINVAL) {
833         switch(errno) {
834             case EUSERS:              有太多登录用户数了
835                 fprintf(stderr,
836                     _("Too many users logged on already.\nTry again later.\n"));
837                 break;
838             case EPROCLIM:
839                 fprintf(stderr,
840                     _("You have too many processes running.\n"));
841                 break;
842             default:
843                 perror("quota (Q_SETUID)");
844         }
845         sleepexit(0);    /* %% */
846     }
847     #endif
848
849     /* paranoia... */

```

```
850  #ifdef SHADOW_PWD
851      endspent();
852  #endif
853      endpwent();
854      当运行到这时，常规情况下，运行 login 的是 account 是 root(如果你手工在命令行上输入 login，那另当别论)
855      /* This requires some explanation: As root we may not be able to
856         read the directory of the user if it is on an NFS mounted
857         filesystem. We temporarily set our effective uid to the user-uid
858         making sure that we keep root privs. in the real uid.
859
860         A portable solution would require a fork(), but we rely on Linux
861         having the BSD setreuid() */
862
863      {
864          char tmpstr[MAXPATHLEN];
865          uid_t ruid = getuid();
866          gid_t egid = getegid();
867
868          /* avoid snprintf - old systems do not have it, or worse,
869             have a libc in which snprintf is the same as sprintf */
870          if (strlen(pwd->pw_dir) + sizeof(_PATH_HUSHLOGIN) + 2 > MAXPATHLEN)
871              quietlog = 0;
872          else {
873              在 home 目录下查找.hushlogin 文件
874              sprintf(tmpstr, "%s/%s", pwd->pw_dir, _PATH_HUSHLOGIN);
875              setregid(-1, pwd->pw_gid);
```



```

875         setreuid(0, pwd->pw_uid);
876         quietlog = (access(tmpstr, R_OK) == 0);           检查 home 目录下有无.hushlogin 文件。然后置是否安静登录标志
877         setuid(0); /* setreuid doesn't do it alone! */
878         setreuid(ruid, 0);
879         setregid(-1, egid);
880     }
881 }
882
883 /* for linux, write entries in utmp and wtmp */

```

utmp 文件用于记录当前系统用户是哪些人及相关信息，这些信息由下面的结构表示

```

struct utmp {
    short ut_type;           /* type of login */
    pid_t ut_pid;           /* PID of login process */
    char ut_line[UT_LINESIZE]; /* device name of tty - "/dev/" */
    char ut_id[4];          /* init id or abbrev. ttyname */
    char ut_user[UT_NAMESIZE]; /* user name */
    char ut_host[UT_HOSTSIZE]; /* hostname for remote login */
    struct exit_status ut_exit; /* The exit status of a process
                                marked as DEAD_PROCESS */

    /* The ut_session and ut_tv fields must be the same size when
       compiled 32- and 64-bit. This allows data files and shared
       memory to be shared between 32- and 64-bit applications */
#ifdef __WORDSIZE == 64 && defined __WORDSIZE_COMPAT32

```

```

    int32_t ut_session;        /* Session ID, used for windowing */
    struct {
        int32_t tv_sec;        /* Seconds */
        int32_t tv_usec;      /* Microseconds */
    } ut_tv;                  /* Time entry was made */
#else
    long int ut_session;       /* Session ID, used for windowing */
    struct timeval ut_tv;      /* Time entry was made */
#endif

    int32_t ut_addr_v6[4];     /* IP address of remote host */
    char __unused[20];         /* Reserved for future use */
};

```

```

884     {
885         struct utmp ut;
886         struct utmp *utp;
887
888         utmpname(_PATH_UTMP);    设置 utmp 的文件路径为/var/run/utmp
889         setutent();
890
891         /* Find pid in utmp.
892         login sometimes overwrites the runlevel entry in /var/run/utmp,
893         confusing sysvinit. I added a test for the entry type, and the problem
894         was gone. (In a runlevel entry, st_pid is not really a pid but some number
895         calculated from the previous and current runlevel).

```

```
896 Michael Riepe <michael@stud.uni-hannover.de>
```

```
897     */
```

下面就是填充这些信息，然后写入 utmp 文件

```
898     while ((utp = getutent()))
```

```
899         if (utp->ut_pid == pid
```

```
900             && utp->ut_type >= INIT_PROCESS
```

```
901             && utp->ut_type <= DEAD_PROCESS)
```

```
902             break;
```

```
903
```

```
904     /* If we can't find a pre-existing entry by pid, try by line.
```

```
905     BSD network daemons may rely on this. (anonymous) */
```

```
906     if (utp == NULL) {
```

```
907         setutent();
```

```
908         ut.ut_type = LOGIN_PROCESS;
```

```
909         strncpy(ut.ut_line, tty_name, sizeof(ut.ut_line));
```

```
910         utp = getutline(&ut);
```

```
911     }
```

```
912
```

```
913     if (utp) {
```

```
914         memcpy(&ut, utp, sizeof(ut));
```

```
915     } else {
```

```
916         /* some gettys/telnetds don't initialize utmp... */
```

```
917         memset(&ut, 0, sizeof(ut));
```

```
918     }
```

```
919
920     if (ut.ut_id[0] == 0)
921         strncpy(ut.ut_id, tty_number, sizeof(ut.ut_id));
922
923         strncpy(ut.ut_user, username, sizeof(ut.ut_user));
924         xstrncpy(ut.ut_line, tty_name, sizeof(ut.ut_line));
925 #ifdef _HAVE_UT_TV          /* in <utmpbits.h> included by <utmp.h> */
926     gettimeofday(&ut.ut_tv, NULL);
927 #else
928     {
929         time_t t;
930         time(&t);
931         ut.ut_time = t;  /* ut_time is not always a time_t */
932         /* glibc2 #defines it as ut_tv.tv_sec */
933     }
934 #endif
935     ut.ut_type = USER_PROCESS;
936     ut.ut_pid = pid;
937     if (hostname) {
938         xstrncpy(ut.ut_host, hostname, sizeof(ut.ut_host));
939         if (hostaddress[0])
940             memcpy(&ut.ut_addr, hostaddress, sizeof(ut.ut_addr));
941     }
942
943     pututline(&ut);
944     endutent();
```

```
945
946  #ifdef HAVE_updwtmp
947      updwtmp(_PATH_WTMP, &ut);
948  #else
949  #if 0
950      /* The O_APPEND open() flag should be enough to guarantee
951         atomic writes at end of file. */
952      {
953          int wtmp;
954
955          if((wtmp = open(_PATH_WTMP, O_APPEND|O_WRONLY)) >= 0) {
956              write(wtmp, (char *)&ut, sizeof(ut));
957              close(wtmp);
958          }
959      }
960  #else
961      /* Probably all this locking below is just nonsense,
962         and the short version is OK as well. */
963      {
964          int lf, wtmp;
965          if ((lf = open(_PATH_WTMPLOCK, O_CREAT|O_WRONLY, 0660)) >= 0) {
966              flock(lf, LOCK_EX);
967              if ((wtmp = open(_PATH_WTMP, O_APPEND|O_WRONLY)) >= 0) {
968                  write(wtmp, (char *)&ut, sizeof(ut));
969                  close(wtmp);
970              }
```

wtmp 的路径为/var/log/wtmp

```

971         flock(lf, LOCK_UN);
972         close(lf);
973     }
974 }
975 #endif
976 #endif
977 }
978
979     dolastlog(quietlog);           report the most recent login of all users or of a given user
980                                   /var/log/lastlog
981     chown(ttyn, pwd->pw_uid,      改变当前 login 用到的终端的 owner 为登录用户所有
982         (gr = getgrnam(TTYGRPNAME)) ? gr->gr_gid : pwd->pw_gid);
983     chmod(ttyn, TTY_MODE);
984
985 #ifdef CHOWNVCS
986     /* if tty is one of the VC's then change owner and mode of the
987        special /dev/vcs devices as well */
988     if (consoletty(0)) {
989         chown(vcsn, pwd->pw_uid, (gr ? gr->gr_gid : pwd->pw_gid));
990         chown(vcsan, pwd->pw_uid, (gr ? gr->gr_gid : pwd->pw_gid));
991         chmod(vcsn, TTY_MODE);
992         chmod(vcsan, TTY_MODE);
993     }
994 #endif
995
996     setgid(pwd->pw_gid);           改为登录用户的 GID

```

```

997
998 #ifdef HAVE_QUOTA
999     quota(Q_DOWARN, pwd->pw_uid, (dev_t)-1, 0);
1000 #endif
1001
1002     if (*pwd->pw_shell == '\0')                如果在/etc/passwd 中没有制定登录 shell , 则设为/bin/sh
1003         pwd->pw_shell = _PATH_BSHELL;
1004
1005     /* preserve TERM even without -p flag */
1006     {
1007         char *ep;
1008
1009         if(!((ep = getenv("TERM")) && (termenv = strdup(ep))))
1010             termenv = "dumb";
1011     }
1012
1013     /* destroy environment unless user has requested preservation */
1014     if (!pflag)
1015     {
1016         environ = (char**)malloc(sizeof(char*));
1017         memset(environ, 0, sizeof(char*));
1018     }
1019     设置如下环境变量 ( HOME , SHELL , TERM , LOGNAME 等 )

```

当我们敲入 env 后看到的环境变量有些就是在这里设的

```

1020     setenv("HOME", pwd->pw_dir, 0);      /* legal to override */
1021     if(pwd->pw_uid)
1022         setenv("PATH", _PATH_DEFPATH, 1);      /usr/local/bin:/bin:/usr/bin
1023     else
1024         setenv("PATH", _PATH_DEFPATH_ROOT, 1); 对 root 而言
1025                                             /usr/local/sbin:/usr/local/bin:/sbin:/bin: /usr/sbin:/usr/bin
1026     setenv("SHELL", pwd->pw_shell, 1);      登录 shell 的路径
1027     setenv("TERM", termenv, 1);
1028
1029     /* mailx will give a funny error msg if you forget this one */
1030     {
1031         char tmp[MAXPATHLEN];
1032         /* avoid snprintf */
1033         if (sizeof(_PATH_MAILDIR) + strlen(pwd->pw_name) + 1 < MAXPATHLEN) {
1034             sprintf(tmp, "%s/%s", _PATH_MAILDIR, pwd->pw_name);
1035             setenv("MAIL", tmp, 0);      用户的信箱在 “ /spool/mail/用户名 ” 下面
1036         }
1037     }
1038
1039     /* LOGNAME is not documented in login(1) but
1040        HP-UX 6.5 does it. We'll not allow modifying it.
1041        */
1042     setenv("LOGNAME", pwd->pw_name, 1);
1043
1044     #ifdef USE_PAM
1045     {

```



```
1046     int i;
1047     char ** env = pam_getenvlist(pamh);
1048
1049     if (env != NULL) {
1050         for (i=0; env[i]; i++) {
1051             putenv(env[i]);
1052             /* D(("env[%d] = %s", i,env[i])); */
1053         }
1054     }
1055 }
1056 #endif
1057
1058 #ifdef DO_PS_FIDDLING
1059     setproctitle("login", username);
1060 #endif
1061
1062     if (!strcmp(tty_name, "ttyS", 4))
1063         syslog(LOG_INFO, _("DIALUP AT %s BY %s"), tty_name, pwd->pw_name);
1064
1065     /* allow tracking of good logins.
1066        -steve philp (sphilp@mail.alliance.net) */
1067
1068     if (pwd->pw_uid == 0) {
1069         if (hostname)
1070             syslog(LOG_NOTICE, _("ROOT LOGIN ON %s FROM %s"),
1071                 tty_name, hostname);
```

```

1072     else
1073         syslog(LOG_NOTICE, _("ROOT LOGIN ON %s"), tty_name);
1074     } else {
1075         if (hostname)
1076             syslog(LOG_INFO, _("LOGIN ON %s BY %s FROM %s"), tty_name,
1077                 pwd->pw_name, hostname);
1078         else
1079             syslog(LOG_INFO, _("LOGIN ON %s BY %s"), tty_name,
1080                 pwd->pw_name);
1081     }
1082
1083     if (!quietlog) {
1084         motd();
1085
1086         #ifdef DO_STAT_MAIL
1087             /*
1088              * This turns out to be a bad idea: when the mail spool
1089              * is NFS mounted, and the NFS connection hangs, the
1090              * login hangs, even root cannot login.
1091              * Checking for mail should be done from the shell.
1092              */
1093             {
1094                 struct stat st;
1095                 char *mail;
1096
1097                 mail = getenv("MAIL");

```

如果 home 目录下无 .hushlogin 文件，则表示正常登录（非安静登录）

所谓的安静登录指的是，登录时不再提示邮件mail，不再显示最后一次登录时间，不输出任何消息

mod()读取/etc/motd 文件中的内容显示，一般为欢迎词等

检查有无新 mail，有则提示

检查在登录用户的信箱里有无新 mail

```
1098         if (mail && stat(mail, &st) == 0 && st.st_size != 0) {
1099             if (st.st_mtime > st.st_atime)
1100                 printf(_("You have new mail.\n"));
1101             else
1102                 printf(_("You have mail.\n"));
1103         }
1104     }
1105 #endif
1106 }
1107
1108     signal(SIGALRM, SIG_DFL);
1109     signal(SIGQUIT, SIG_DFL);
1110     signal(SIGTSTP, SIG_IGN);
1111
1112 #ifdef USE_PAM
1113     /*
1114      * We must fork before setuid() because we need to call
1115      * pam_close_session() as root.
1116      */
1117
1118     childPid = fork();          login process 孕育新进程
1119     if (childPid < 0) {
1120         int errsv = errno;
1121         /* error in fork() */
1122         fprintf(stderr, _("login: failure forking: %s"), strerror(errsv));
1123         PAM_END;
```

```
1124     exit(0);
1125 }
1126
1127 if (childPid) {          login 父进程
1128     /* parent - wait for child to finish, then cleanup session */
1129     signal(SIGHUP, SIG_IGN);
1130     signal(SIGINT, SIG_IGN);
1131     signal(SIGQUIT, SIG_IGN);
1132     signal(SIGTSTP, SIG_IGN);
1133     signal(SIGTTIN, SIG_IGN);
1134     signal(SIGTTOU, SIG_IGN);
1135
1136     wait(NULL);          login 父进程就是 suspend , 等待子进程的结束
1137     PAM_END;
1138     exit(0);
1139 }
1140
1141 /* child */              下面的代码在 login 的子进程中执行
1142 /*
1143  * Problem: if the user's shell is a shell like ash that doesnt do
1144  * setsid() or setpgrp(), then a ctrl-\, sending SIGQUIT to every
1145  * process in the pgrp, will kill us.
1146  */
1147
1148 /* start new session */
1149 setsid();
```

```

1150
1151     /* make sure we have a controlling tty */
1152     opentty(ttyn);           把 login 子进程的 file handle 0,1,2 指向父进程的 tty 设备 (即父子进程公有该 tty 设备)
1153     openlog("login", LOG_ODELAY, LOG_AUTHPRIV);  /* reopen */
1154
1155     /*
1156      * TIOCSCTTY: steal tty from other process group.
1157      */
1158     if (ioctl(0, TIOCSCTTY, 1))
1159         syslog(LOG_ERR, _("TIOCSCTTY failed: %m"));
1160 #endif
1161     signal(SIGINT, SIG_DFL);
1162
1163     /* discard permissions last so can't get killed and drop core */
1164     if(setuid(pwd->pw_uid) < 0 && pwd->pw_uid) {           子进程的 UID 与 GID 还为 root, 现在把它设成登录用户的 UID 与 GID
1165         syslog(LOG_ALERT, _("setuid() failed"));
1166         exit(1);
1167     }
1168
1169     /* wait until here to change directory! */
1170     if (chdir(pwd->pw_dir) < 0) {                           把当前目录切换到登录用户的 home 目录
1171         printf(_("No directory %s!\n"), pwd->pw_dir);
1172         if (chdir("/"))                                     如果登录用户没有设 home 目录, 则设为根目录
1173             exit(0);
1174         pwd->pw_dir = "/";
1175         printf(_("Logging in with home = %s\n"), pwd->pw_dir);

```

```
1176     }
1177     下面就是 login 进程的最后一步，就是执行用户的登录 shell
1178     /* if the shell field has a space: treat it like a shell script */
1179     if (strchr(pwd->pw_shell, ' ')) {      如果指向登录 shell 的字符串中有空格
1180         buff = malloc(strlen(pwd->pw_shell) + 6);
1181
1182         if (!buff) {
1183             fprintf(stderr, _("login: no memory for shell script.\n"));
1184             exit(0);
1185         }
1186
1187         strcpy(buff, "exec ");
1188         strcat(buff, pwd->pw_shell);
1189         childArgv[childArgc++] = "/bin/sh";
1190         childArgv[childArgc++] = "-sh";
1191         childArgv[childArgc++] = "-c";
1192         childArgv[childArgc++] = buff;      “ /bin/sh -sh -c exec pwd->pw_shell ”
1193     } else {
1194         tbuf[0] = '-';
1195         xstrncpy(tbuf + 1, ((p = rindex(pwd->pw_shell, '/')) ?
1196             p + 1 : pwd->pw_shell),
1197             sizeof(tbuf)-1);
1198
1199         childArgv[childArgc++] = pwd->pw_shell;
1200         childArgv[childArgc++] = tbuf;      “ pwd->pw_shell ”
1201     }
```

```
1202
1203     childArgv[childArgc++] = NULL;
1204
1205     execvp(childArgv[0], childArgv + 1);      login 子进程执行登录 shell
1206
1207     errsv = errno;                            不应该在执行到这儿，否则那就是错了
1208
1209     if (!strcmp(childArgv[0], "/bin/sh"))
1210         fprintf(stderr, _("login: couldn't exec shell script: %s.\n"),
1211             strerror(errsv));
1212     else
1213         fprintf(stderr, _("login: no shell: %s.\n"), strerror(errsv));
1214
1215     exit(0);
1216 }
1217
1218 #ifndef USE_PAM
1219 static void
1220 getloginname(void) {                          向用户询问登录名
1221     int ch, cnt, cnt2;
1222     char *p;
1223     static char nbuf[UT_NAMESIZE + 1];
1224
1225     cnt2 = 0;
1226     for (;;) {
1227         cnt = 0;
```

```
1228     printf(_("\n%s login: "), thishost); fflush(stdout);
1229     for (p = nbuf; (ch = getchar()) != '\n'; ) {
1230         if (ch == EOF) {
1231             badlogin("EOF");
1232             exit(0);
1233         }
1234         if (p < nbuf + UT_NAMESIZE)
1235             *p++ = ch;
1236
1237         cnt++;
1238         if (cnt > UT_NAMESIZE + 20) {
1239             fprintf(stderr, _("login name much too long.\n"));
1240             badlogin(_("NAME too long"));
1241             exit(0);
1242         }
1243     }
1244     if (p > nbuf) {
1245         if (nbuf[0] == '-')
1246             fprintf(stderr,
1247                 _("login names may not start with '-'.\n"));
1248         else {
1249             *p = '\0';
1250             username = nbuf;
1251             break;
1252         }
1253     }
```

登录名由全局变量指向


```
1254
1255     cnt2++;
1256     if (cnt2 > 50) {
1257         fprintf(stderr, _("too many bare linefeeds.\n"));
1258         badlogin(_("EXCESSIVE linefeeds"));
1259         exit(0);
1260     }
1261 }
1262 }
1263 #endif
1264
1265 /*
1266  * Robert Ambrose writes:
1267  * A couple of my users have a problem with login processes hanging around
1268  * soaking up pts's. What they seem to hung up on is trying to write out the
1269  * message 'Login timed out after %d seconds' when the connection has already
1270  * been dropped.
1271  * What I did was add a second timeout while trying to write the message so
1272  * the process just exits if the second timeout expires.
1273  */
1274
1275 static void
1276 timedout2(int sig) {
1277     struct termio ti;
1278
1279     /* reset echo */
```

```
1280     ioctl(0, TCGETA, &ti);
1281     ti.c_lflag |= ECHO;
1282     ioctl(0, TCSETA, &ti);
1283     exit(0);          /* %% */
1284 }
1285
1286 static void
1287 timedout(int sig) {
1288     signal(SIGALRM, timedout2);
1289     alarm(10);
1290     fprintf(stderr, _("Login timed out after %d seconds\n"), timeout);
1291     signal(SIGALRM, SIG_IGN);
1292     alarm(0);
1293     timedout2(0);
1294 }
1295
1296 #ifndef USE_PAM
1297 int
1298 rootterm(char * ttyn)
1299 {
1300     int fd;
1301     char buf[100],*p;
1302     int cnt, more = 0;
1303
1304     fd = open(SECURETTY, O_RDONLY);
1305     if(fd < 0) return 1;
```

```
1306
1307     /* read each line in /etc/securetty, if a line matches our ttyline
1308        then root is allowed to login on this tty, and we should return
1309        true. */
1310     for(;;) {
1311         p = buf; cnt = 100;
1312         while(--cnt >= 0 && (more = read(fd, p, 1)) == 1 && *p != '\n') p++;
1313         if(more && *p == '\n') {
1314             *p = '\0';
1315             if(!strcmp(buf, ttyn)) {
1316                 close(fd);
1317                 return 1;
1318             } else
1319                 continue;
1320         } else {
1321             close(fd);
1322             return 0;
1323         }
1324     }
1325 }
1326 #endif /* !USE_PAM */
1327
1328 jmp_buf motdinterrupt;
1329
1330 void
1331 motd(void) {
```

```
1332     int fd, nchars;
1333     void (*oldint)(int);
1334     char tbuf[8192];
1335
1336     if ((fd = open(_PATH_MOTDFILE, O_RDONLY, 0)) < 0)      打开登录欢迎文件/etc/motd
1337         return;
1338     oldint = signal(SIGINT, sigint);
1339     if (setjmp(motdinterrupt) == 0)
1340         while ((nchars = read(fd, tbuf, sizeof(tbuf))) > 0)  输出该欢迎文件中的内容
1341             write(fileno(stdout), tbuf, nchars);
1342     signal(SIGINT, oldint);
1343     close(fd);
1344 }
1345
1346 void
1347 sigint(int sig) {
1348     longjmp(motdinterrupt, 1);
1349 }
1350
1351 #ifndef USE_PAM          /* PAM takes care of this */
1352 void
1353 checknologin(void) {
1354     int fd, nchars;
1355     char tbuf[8192];
1356
1357     if ((fd = open(_PATH_NOLOGIN, O_RDONLY, 0)) >= 0) {      打开/etc/nologin 文件
```

[illegible]

```
1384         else
1385             printf(_("on %.*s\n"),
1386                 (int)sizeof(ll.ll_line), ll.ll_line);
1387     }
1388     lseek(fd, (off_t)pwd->pw_uid * sizeof(ll), SEEK_SET);
1389 }
1390 memset((char *)&ll, 0, sizeof(ll));
1391 time(&ll.ll_time);
1392 xstrncpy(ll.ll_line, tty_name, sizeof(ll.ll_line));
1393 if (hostname)
1394     xstrncpy(ll.ll_host, hostname, sizeof(ll.ll_host));
1395
1396 write(fd, (char *)&ll, sizeof(ll));
1397 close(fd);
1398 }
1399 }
1400
1401 void
1402 badlogin(const char *name) {
1403     if (failures == 1) {
1404         if (hostname)
1405             syslog(LOG_NOTICE, _("LOGIN FAILURE FROM %s, %s"),
1406                 hostname, name);
1407         else
1408             syslog(LOG_NOTICE, _("LOGIN FAILURE ON %s, %s"),
1409                 tty_name, name);
```

```
1410     } else {
1411     if (hostname)
1412         syslog(LOG_NOTICE, _("%d LOGIN FAILURES FROM %s, %s"),
1413             failures, hostname, name);
1414     else
1415         syslog(LOG_NOTICE, _("%d LOGIN FAILURES ON %s, %s"),
1416             failures, tty_name, name);
1417     }
1418 }
1419
1420 /* Should not be called from PAM code... */
1421 void
1422 sleepexit(int eval) {
1423     sleep(SLEEP_EXIT_TIMEOUT);
1424     exit(eval);
1425 }
```

环境

login 源码摘自 util-linux-2.12r package

联系

Walter Zhou

<mailto:z-l-dragon@hotmail.com>