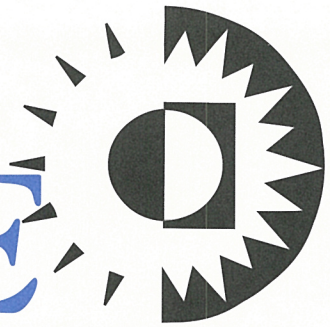


INSIDE SOLARIS™

Tips & techniques for users of SunSoft Solaris



IN THIS ISSUE

1
Understanding the /proc
file system

4
Exploring the secrets of
snoop

8
Working with PGP, part 2

14
"Rotating" files

15
Using CDE

Understanding the /proc file system

by Robert Owen Thomas

On more than one occasion, I've been asked about the /proc file system. What is it? What are all those huge files doing there? Can I delete them? In this article, we'll detail the /proc file system, describe some tools you can use to take advantage of the power of /proc, and step through a sample program that will demonstrate how easily a systems administrator can interact with /proc.

Introducing /proc

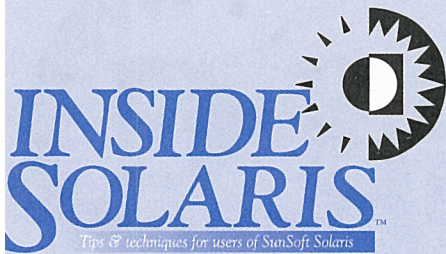
In the bad old days, getting at process data (e.g., running the **ps(1)** command) required direct access to kernel memory. Gaining this access was horribly complex and required superuser authority. AT&T solved the process data access problem by releasing UNIX SVR4. Now, you can access process data simply by entering /proc.

The /proc file system isn't a file system in the standard sense. Rather, the /proc file system is an interface to the address space of running processes. With /proc, you can use standard UNIX system calls (e.g., **open()**, **read()**, **write()**, and **ioctl()**) to query or manipulate the processes' address space. In fact, the Solaris **ps(1)** com-

mand uses /proc to determine the status of the processes listed in the process table.

The large files found in /proc are the address spaces of running processes—not standard UNIX files. Each filename is, in fact, just the PID of the running processes. The owner and group owner of each file are the real UID and primary group of the process's owner. The permission bits control access, as with any UNIX file. The most confusing part—the file's size—is actually quite simple: It's the total amount of memory (image size) the process uses. Since this amount doesn't represent actual bytes on disk, you're not losing precious disk space to the files under /proc. So don't delete those files! Take a look at the sample /proc listing in **Figure A** on the next page.

The method for manipulating files under /proc is the same as for ordinary UNIX files. All of your favorite system calls will work, including **ioctl()**. In the kernel, the vnode operations performed on a file under /proc are for procf's. This means that the actual vnode access system calls (e.g., **lookuppn()**) are eventually passed to procf's-savvy system calls (e.g., **prlookup()**).



Inside Solaris (ISSN 1081-3314) is published monthly by The Cobb Group.

Prices
 U.S. \$115/yr (\$11.50 each)
 Outside U.S. \$135/yr (\$16.95 each)

Phone and Fax
 US toll free (800) 223-8720
 Local (502) 493-3300
 Customer Relations fax (502) 491-8050
 Editorial Department fax (502) 491-4200

Address
 Send your tips, special requests, and other correspondence to

The Editor, *Inside Solaris*
 9420 Bunsen Parkway, Suite 300
 Louisville, KY 40220
 Internet: inside_solaris@zd.com.

For subscriptions, fulfillment questions, and requests for group subscriptions, address your letters to

Customer Relations
 9420 Bunsen Parkway, Suite 300
 Louisville, KY 40220
 Internet: cobb_customer_relations@zd.com

Staff
 Editor-in-Chief Garrett Suhm
 Contributing Editors Marc Schwarz
 Don Kuenz
 Lance E. Spitzner
 Robert Owen Thomas
 Werner Klausner
 Print Designer Lance Teitworth
 Editors Karen S. Shields
 Jill Suhm
 Joan McKim
 Michael E. Jones
 Publications Coordinator Linda Recktenwald
 Managing Editor Kent Michels
 Associate Publisher Michael Stephens
 Circulation Manager Mike Schroeder
 Publisher Jon Pyles

Back Issues
 To order back issues, call Customer Relations at (800) 223-8720. Back issues cost \$11.50 each, \$16.95 outside the US. We accept MasterCard, Visa, or American Express.

Postmaster
 Periodicals postage paid in Louisville, KY.
 Postmaster: Send address changes to

Inside Solaris
 P.O. Box 35160
 Louisville, KY 40232

Copyright
 Copyright © 1998, The Cobb Group, a division of Ziff-Davis Inc. The Cobb Group and ZD Journals are registered trademarks of Ziff-Davis Inc. All rights reserved. Reproduction in whole or in part in any form or medium without express written permission of Ziff-Davis is prohibited. The Cobb Group reserves the right, with respect to submissions, to revise, republish, and authorize its readers to use the tips submitted for personal and commercial use. Information furnished in this newsletter is believed to be accurate and reliable; however, no responsibility is assumed for inaccuracies or for the information's use.

Inside Solaris is a trademark of Ziff-Davis Inc. Sun, Sun Microsystems, the Sun logo, SunSoft, the SunSoft logo, Solaris, SunOS, SunInstall, OpenBoot, OpenWindows, DeskSet, ONC, and NFS are trademarks or registered trademarks of Sun Microsystems, Inc. UNIX and OPEN LOOK are registered trademarks of UNIX System Laboratories, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

What can /proc tell me?

The tools bundled with Solaris that use /proc are quite informative. Found under /usr/proc/bin, the tools provide a handy way of accessing critical bits of data about a given process. For example, let's say you want to know how many files a process has open. You could use `crash(1M)`, but you don't have root! No fear, you have /usr/proc/bin/pfiles to help you. **Figure B** on page 4 shows a sample use of the `pfiles(1)` command.

As shown in the code, using any of the /proc commands is as simple as the command name followed by the PID. *Keep those permission bits in mind, however!* As with all UNIX files, you'll be unable to access the process data for a given PID if you don't have the proper permissions.

Spend some time with the `proc(1)` man page and familiarize yourself with the commands detailed therein. You'll find ways to list the libraries used by a process, the signal disposition for a process, and the process's credentials—you can even stop and restart processes!

Writing /proc tools

The real beauty of /proc is that anything you can possibly want to know about a process is there; you just ask for it. Two structures, `prstatus` and `prpsinfo`, both of which are defined in /usr/include/sys/procfs.h, can be populated with a wealth of information about a process.

Here's one example. A developer wants to know just how much memory his application is using. Simple: use `ls/proc/<PID>`. However, he also wants to see much more

Figure A

```
: natasha; ls -l /proc
total 43384
-rw----- 1 root  root  0 Apr 2 20:07 00000
-rw----- 1 root  root 393216 Apr 2 20:07 00001
-rw----- 1 root  root  0 Apr 2 20:07 00002
-rw----- 1 root  root  0 Apr 2 20:07 00003
-rw----- 1 root  root 1695744 Apr 2 20:07 00081
-rw----- 1 root  root 1597440 Apr 2 20:07 00083
-rw----- 1 root  root 1777664 Apr 2 20:08 00096
-rw----- 1 root  root 1683456 Apr 2 20:08 00099
-rw----- 1 root  root 1589248 Apr 2 20:08 00101
-rw----- 1 root  root 1445888 Apr 2 20:08 00116
-rw----- 1 root  root 1404928 Apr 2 20:08 00126
-rw----- 1 root  root  798720 Apr 2 20:08 00135
-rw----- 1 root  root 1368064 Apr 2 20:08 00195
-rw----- 1 root  root 1585152 Apr 2 20:08 00197
-rw----- 1 root  root 1368064 Apr 2 20:08 00200
-rw----- 1 root  other 225280 Apr 2 20:08 00201
-rw----- 1 root  root 1454080 Apr 2 20:08 00203
-rw----- 1 root  root 1519616 Apr 2 20:14 00243
-rw----- 1 rthomas wheel 1499136 Apr 2 20:14 00245
-rw----- 1 rthomas wheel  806912 Apr 2 20:16 00261
: natasha;
```

You can see from this sample listing that you don't lose precious disk space to the files under /proc.

Listing A: The source code to memlook

```
/*
 * @(#)memlook.c 1.0 10 Nov 1997
 * Robert Owen Thomas robt@cymru.com
 * memlook.c -- A process memory
 * utilization reporting tool.
 *
 */

static char copyright[] = "@(#)memlook.c 1.0
10 Nov 1997 Robert Owen Thomas
robt@cymru.com";

#include <stdio.h>
#include <sys/types.h>
#include <sys/signal.h>
#include <sys/syscall.h>
#include <sys/procfs.h>
#include <sys/param.h>
#include <fcntl.h>

int counter = 10;

int showUsage(const char *);
void getInfo(int, int);

int
main (int argc, char *argv[])
{
    int fd,
        pid,
        timeloop = 0;
    char pidpath[BUFSIZ];

    switch (argc) {
    case 2:
        break;
    case 3:
        timeloop = atoi(argv[2]);
        break;
    default:
        showUsage(argv[0]);
        break;
    }

    pid = atoi(argv[1]);
    sprintf(pidpath, "/proc/%-d", pid);

    if ((fd = open(pidpath, O_RDONLY)) < 0) {
        perror(pidpath);
        exit(1);
    }

    if (0 < timeloop) {
        for (;;) {
            getInfo(fd, pid);
            sleep(timeloop);
        }

        getInfo(fd, pid);
        close(fd);
        exit(0);
    }

    int
    showUsage(const char *programe)
    {
        fprintf(stderr, "%s: usage: %s <PID>
        ↪[time delay]\n", programe, programe);
        exit(3);
    }

    void
    getInfo(int fd, int pid)
    {
        prpsinfo_t prp;
        prstatus_t prs;

        if (ioctl(fd, PIOCPSTATUS, &prp) < 0) {
            perror("ioctl");
            exit(5);
        }

        if (ioctl(fd, PIOCPSTATUS, &prp) < 0) {
            perror("ioctl");
            exit(7);
        }

        if (counter > 9) {
            fprintf(stdout, "PID\tIMAGE\t\tRSS\t\tHEAP\t\tSTACK\n");
            counter = 0;
        }

        fprintf(stdout, "%d\t%-9d\t%-9d\t%-15d\t%-15d\n",
        ↪pid, prp.pr_bysize, prp.pr_byrssize, prs.pr_brksize,
        ↪prs.pr_stksize);

        counter++;
    }
}
```


detail. He needs to see the total image size, the resident set size, the heap size, and the stack size. Further, he wishes to track this data over time in a manner similar to **vmstat(1M)**. All told, a seemingly daunting task!

Figure B

```
: natasha; ps
  PID TTY      TIME CMD
  245 pts/0    0:00 ksh
: natasha; pfiles 245
245:    -ksh
Current rlimit: 64 file descriptors
  0: S_IFCHR mode:0620 dev:32,24 ino:197800 uid:101 gid:7 rdev:24,0
    O_RDWR
  1: S_IFCHR mode:0620 dev:32,24 ino:197800 uid:101 gid:7 rdev:24,0
    O_RDWR
  2: S_IFCHR mode:0620 dev:32,24 ino:197800 uid:101 gid:7 rdev:24,0
    O_RDWR
 63: S_IFREG mode:0600 dev:32,27 ino:2881 uid:101 gid:666 size:282
    O_RDWR|O_APPEND close-on-exec
: natasha;
```

You can use the `pfiles(1)` command to know how many files a process has open.

Figure C

```
: natasha; memlook 245
PID    IMAGE      RSS      HEAP      STACK
245    1499136    1044480  24581     8192
: natasha;
```

Here's a sample output from `memlook`.

However, we can simplify this programmatic challenge by using the `/proc` file system. The tool called **memlook** will display memory statistics for a given PID. You can also give **memlook** an interval, which will cause it to recheck the memory usage at a specified interval of seconds. This capability is particularly handy for trend analysis. **Figure C** shows a sample output, and **Listing A** on page 3 contains the source code.

Feel free to experiment with **memlook!** Peruse the **prstatus** and **prpsinfo** structures that are found in `/usr/include/sys/procfs.h`, and pick out some values you or your user community might find useful. You should avoid using **write()** or **ioctl()** with the `/proc` file system until you really master the art. Random **write()** calls into a process will leave the process grumpy, at best.

Conclusion

The `/proc` file system is a powerful ally when you're debugging a thorny system issue or attempting to obtain the status of a given process. By using the structures provided within Solaris, you can create powerful yet simple tools that will provide rich detail you need to complete your task. ❖

Rob Thomas is an aspiring blues guitarist who earns his living with UNIX and networking. You can reach him at robt@cymru.com and <http://www.cymru.com/~robt>.

NETWORK SECURITY

Exploring the secrets of snoop

by Lance Spitzner

The use of sniffers has exploded in popularity over the past several years. The tools—from Network General's Netxray and Microsoft's Network Monitor to public domain tools, such as Etherman and Curry Sniffer—are readily available. You can use these tools for various purposes, including network troubleshooting, traffic analysis, and node discovery.

In this article, we'll discuss one of the most common, yet effective, sniffers—`snoop`. Of all the

sniffers, `snoop` is one standby that you always have access to with Solaris. We'll demonstrate how to leverage `snoop`, providing examples focusing on network security.

What is snoop?

`Snoop` is an executable binary that puts your system's interface(s) in promiscuous mode. Promiscuous mode enables `snoop` to accept all

packets on your network, either in real time or capture file format. To run snoop, you must be root. What makes snoop so powerful is the detail of information it provides and the flexibility of the tool itself.

As an introduction, we'll focus on snoop commands and how to get the information we want. Then, we'll look at analyzing network traffic with real-world examples, focusing on security. The examples will be IP, but you can use snoop to capture and analyze other network packets, such as DECnet and AppleTalk. For packet analysis, we'll use the standard seven-layer OSI model, as shown in **Table A**.

The OSI (Open Systems Interconnection) model was developed in 1974 by the International Standards Organization. The seven-layer model is an international standard that allows systems to communicate with each other as if they were the same system. Each layer has a specific purpose independent of the others, yet each layer "talks" to the next layer.

A packet starts at the application layer, works its way down the stack, and is then sent to the other system. The other system receives the packet at the first layer, then sends it back up the stack. Not all layers may be used, specifically layers 5 and 6.

Using snoop

First, you decide whether you want real-time data or to capture packets to a snoop-capture file. Most of the time you'll capture the data to a file. In real-time mode, the data flies across your screen much too fast for you to read. The only real benefit of real-time mode is to give you a quick feel for the traffic that's moving on your network. To do some serious analysis, you'll

want to capture packets to a file so you can take your time with them.

To capture data to a file, type

```
#snoop -o filename
```

This command saves all the data in binary format to *filename*. To see data in real-time, exclude the **-o filename** command. Otherwise, all command syntax is the same for snoop.

Next, we determine how many packets to capture. If the number of packets isn't specified, snoop will continue to gather packets until you hit [Ctrl]C or run out of resources. To set the number, type

```
#snoop -o filename -c 1000
```

With this command, snoop will capture 1,000 packets in about 60 seconds on a standard 10Mbps network.

At this point, we want to decide what level of detail we need. Snoop comes in three flavors: summary (default), verbose summary (-V), and verbose mode (-v). Of the three, summary gives us the least detail, including only the highest protocol level, layer 5, 6, or 7, and packet source or destination. A single packet in summary mode is shown as follows:

```
27 0.01743 squirrel -> ICARUS.CC.UIC.EDU
    ↳TELNET C port=45330
```

This is the 27th packet captured, showing a Telnet connection between squirrel and my school account. The time between packet 26 and 27 is just 0.01743 of a second.

Verbose summary (-V) gives us all the layers of the OSI model (layers 2, 3, 4, and 5, 6, or 7),

Table A: OSI seven-layer model

| Layer | Description |
|--|--|
| Layer 7 Application (SMTP, TELNET) | Defines the network applications. |
| Layer 6 Presentation (Encryption) | Data translation (format of the data). |
| Layer 5 Session | Establishes, maintains, and disconnects a communications link between two stations on a network. |
| Layer 4 Transport (TCP, UDP) | Provides for end-to-end transmission of data. |
| Layer 3 Network (IP, IPX, AppleTalk) | Controls forwarding of packets between stations. |
| Layer 2 Data Link (Ethernet, Token Ring) | Physical layer addressing, synchronizes transmission and handles frame-level error control and recovery. |
| Layer 1 Physical (UTP, Fiber) | Method used to transmit data (media, voltage, etc). |

but in summarized fashion—one line for each layer. In the following example, we'll see packet 27 again:

```

27 0.01743 squirrel -> ICARUS.CC.UIC.EDU
    ↳ETHER Type=0800 (IP), size = 58 bytes
27 0.01743 squirrel -> ICARUS.CC.UIC.EDU IP
    ↳D=128.248.121.53 S=208.194.41.20 LEN=44,
    ↳ID=6082
27 0.01743 squirrel -> ICARUS.CC.UIC.EDU
    ↳TCP D=23 S=45330 Syn Seq=678057692 Len=0
    ↳Win=8760
27 0.01743 squirrel -> ICARUS.CC.UIC.EDU
    ↳TELNET C port=45330

```

Notice that we get layer 2 (ETHER), layer 3 (IP), layer 4 (TCP), and layer 7 (Telnet). The code also gives Syn and Seq (sequence number). There's no Ack (Acknowledge number), so this is the first packet for this Telnet session.

Verbose gives us all the gory details of each packet, all the way to the bit level on the OSI model. The following lines of code show packet 27 in verbose mode:

```

ETHER: ----- Ether Header -----
ETHER:
ETHER: Packet 27 arrived at 10:40:36.07
ETHER: Packet size = 58 bytes
ETHER: Destination = 8:0:20:8d:fc:d2, Sun
ETHER: Source      = 8:0:20:c:df:aa, Sun
ETHER: Ethertype = 0800 (IP)
ETHER:
IP: ----- IP Header -----
IP:
IP: Version = 4
IP: Header length = 20 bytes
IP: Type of service = 0x00
IP:   xxx. .... = 0 (precedence)
IP:   ...0 .... = normal delay
IP:   .... 0... = normal throughput
IP:   .... .0.. = normal reliability
IP: Total length = 44 bytes
IP: Identification = 6082
IP: Flags = 0x4
IP:   .1.. .... = do not fragment
IP:   ..0. .... = last fragment
IP: Fragment offset = 0 bytes
IP: Time to live = 255 seconds/hops
IP: Protocol = 6 (TCP)
IP: Header checksum = 7005
IP: Source address = 208.194.41.20, squirrel
IP: Destination address = 128.248.121.53,
    ↳ICARUS.CC.UIC.EDU
IP: No options
IP:
TCP: ----- TCP Header -----
TCP:

```

```

TCP: Source port = 45330
TCP: Destination port = 23 (TELNET)
TCP: Sequence number = 678057692
TCP: Acknowledgement number = 0
TCP: Data offset = 24 bytes
TCP: Flags = 0x02
TCP:   ..0. .... = No urgent pointer
TCP:   ...0 .... = No acknowledgement
TCP:   .... 0... = No push
TCP:   .... .0.. = No reset
TCP:   .... ..1. = Syn
TCP:   .... ...0 = No Fin
TCP: Window = 8760
TCP: Checksum = 0x517a
TCP: Urgent pointer = 0
TCP: Options: (4 bytes)
TCP:   - Maximum segment size = 1460 bytes
TCP:
TELNET: ----- TELNET: -----
TELNET:
TELNET: ""
TELNET:

```

Here we see detailed information for each layer, layer 2 (Ethernet), layer 3 (IP), and layer 4 (TCP) header. Go to <http://info.internet.isi.edu:80/in-notes/rfc/files/> to see RFC 894 (Ether), 791 (IP), and 793 (TCP) for specific header information.

No one level of detail is better than the other. It depends on what type of information you're looking for. However, snoop can be resource-intensive. In verbose mode, snoop may even overwhelm the system. Depending on your network traffic, snoop may be forced to drop packets. In some cases, you may have to use a dedicated server for snoop. Choosing this recourse will depend on your verbose level and number of packets gathered.

To read a capture file, use `-i filename`. If you captured packets in verbose mode, you can read a capture file in summary, verbose summary, or verbose mode. You should scan through the capture file in summary mode, identify which packets are interesting, then view specific packets in verbose mode. To look at a specific packet, use `-ppacket#`. This example looks at packets 10-32 and packet 56 in verbose mode:

```
#snoop -i filename -v -p10-32,56
```

Now we're ready to leverage the true power of snoop. Snoop has a variety of filtering tools, allowing us to focus on the type of packets we capture, whether they're source, destination, protocol layer, etc. We'll cover some of the most commonly used options. However, for complete information, be sure to do a **man** on snoop(1).

We'll select which systems will be snooped by their MAC (layer 2) or IP (layer 3). This process limits which packets are captured at the interface.

If you have just one node you want to snoop, include its IP address. If there are several, use the expression **and** or **or** between the nodes. You can focus the expression more precisely with the qualifier **from** or **to**, which matches the source or destination address. The **!** or **not** performs a logical NOT operation. Finally, the expression **net** captures all packets that belong to a specific network. The following command captures all packets coming from zeus going to 8:0:20:f1:b3:51, or packets belonging to the network 192.168.3.0, except 192.168.3.58. Note, the host name zeus must be resolvable, be it /etc/hosts or DNS:

```
snoop -o filename from zeus or to
8:0:20:f1:b3:51 or net 192.168.3.0 not
192.168.3.58
```

Just as we can qualify specific hosts or networks at layer 2 or 3, we can limit packets captured at layers 4, 5, 6, and 7. At layer 4, we can qualify **tcp**, **udp**, or **icmp** (actually RFC 792 states **icmp** is a layer-3 protocol, but I've placed it here to reflect snoop's **man** page). For layers 5, 6, and 7, use the qualifiers **port** and **rpc** (based on the /etc/services and /etc/rpc files). The following command captures all DNS or NFS packets:

```
snoop -o filename -V port domain or rpc nfs
```

Snoop and security

Now we'll apply snoop to your network security. With snoop, you silently sit on the network and capture data. Unlike active measures, such as network discovery using ICMP, snoop doesn't alert anyone to its presence. This stealth allows you to analyze your network's security without notifying anyone. Also, snoop can run over a long period of time, compared to active measures that run at a single point in time. If a server is down for several minutes while you are ping-ing the network, you could miss the event. Snoop will pick up these servers, as long as they eventually send or receive traffic.

Snoop does two critical things for security: It tells you who's on your network and what they're doing. After you've identified your security concern, then you configure snoop to find that information.

Securing the gates

Often, a security concern is having a node or gateway on your network that you don't know about. This node could be an innocent dial-up server or a gateway set up by a hacker. Active measures will tell you who's on the network—*only* if the machine is on.

But what if a node is on only at night or has been configured not to ICMP_REPLY? Using the qualifiers we've covered, snoop could capture specific information about your network. With a PERL or shell script, you could then parse this information, identifying unknown nodes on your network.

Another security issue concerns what's actually happening in your network, specifically what packets are going where. You may be wondering about certain Web sites or downloads. Perhaps you're concerned that users are downloading the latest hacker tools. You can snoop your network, looking for FTP downloads from known Web sites.

Routing and routes are another security issue. Many people consider routing strictly a performance issue, but it also demands security. For example, many local networks configure their systems with a single default router. Imagine your surprise if you discovered a system broadcasting RIP. You thought certain network traffic was going out the default router, when in reality it was being redirected.

RIP itself is also an insecure protocol. RIP packets can be easily spoofed, allowing someone to change your routing tables. Snoop allows you to identify these surprises, identifying the layer-7 protocols (summary mode), and the contents of these packets (verbose mode).

Identifying sources

Perhaps you have several critical servers recently hit with denial of service attacks, such as land.c or ping of death. You can qualify snoop to look for these attacks and potentially find the source. You could identify land.c by capturing packets with the same layer-3 source and destination and maybe find the culprit with layer-2 information. For ping of death, look for **icmp** packets with extremely large lengths.

So far we've discussed what snoop can do; now we'll cover what snoop cannot do. Most sniffers, including snoop, can't operate in a switched environment—unlike active measures. Snoop records only packets that

cross the designated interface. Switches block and forward IP packets based on their MAC, or layer 2, address. If you have a switch, snoop will capture only the packets in its collision domain.

Where you snoop is just as important as *what* you snoop. Always keep collision domains in mind. If you want to monitor all the traffic on your network, place your sniffer on the Internet router segment. This way, you're capturing all Internet traffic, and you aren't limited to specific collision domains.

You can also use snoop's limitation to your advantage. A common tactic hackers use is to compromise a system and implement a sniffer. Once compromised, the sniffer picks up user names and passwords. Several months ago, the SANS Institute was compromised by this method.

Over the firewall

Systems on your DMZ and the network segment between your Internet router and firewall are prime targets. Companies frequently place unsecured systems, such as web servers, outside

the firewall. However, once compromised, these systems make excellent platforms for capturing user names and passwords.

To protect your network, place these systems behind a switch. Then, if they're compromised, they're isolated in their collision domain and can still protect you from sniffing. (You may also want to hardcode the MAC address on the switch to specific ports.)

Conclusion

Snoop is an extremely powerful and flexible tool. Security is just one of the many areas where you can take advantage of snoop. And the best part is, we've only begun to scratch the surface of snoop's capabilities. Stay tuned. ❖

A relative newcomer to the info security world, Lance learns by blowing up UNIX systems at home. Previously, he was a tank officer in the Army's Rapid Deployment Force where he blew up things of a different nature. You can reach him at lspitzner@enteract.com.

PROTECT YOUR PRIVACY

Working with PGP, part 2

by Marc Schwarz

Last month we demonstrated how to install and set up PGP for basic tasks. In this article, we'll discuss a number of PGP's intermediate and advanced features.

Encrypting to multiple recipients

Often, you must send an encrypted message to many people. For example, you might want to send a *Discussion Topics* listing to everyone who was invited to a closed private conference. If the meeting is small, you might think nothing of manually running the

```
pgp -fesa (recipient) -u (yourself) <
    plainfile.txt > encryptedfile.asc
```

command sequence once for every person on the invitation list.

If, however, you're expecting a few hundred people at this meeting, running PGP in

this manner is tedious *and* subject to error. Fortunately, PGP has a built-in feature that lets you encrypt a message for multiple parties using only one command call. Type the command

```
pgp -e file.txt (uname1) (uname2)... (unameN)
```

where **file.txt** is the name of the file you want to encrypt, and **(uname1)**, **(uname2)**, ..., **(unameN)** are the user names of the people who'll receive your message. Any one of these individuals will be able to decrypt your note.

You can combine this command sequence with most of the PGP switch options, such as **-s**, which attaches a signature to the encrypted file. Don't use the **-f** switch, which normally permits UNIX redirection. Including the **-f** option in your multiple-recipient **pgp** command call may hang the system.

Mailing to multiple recipients

While PGP provides an expedient method for generating the encrypted file, you're still left with the task of E-mailing the encrypted note to everyone on your list. Again, if the list is small, then sending the notes via individual calls to the UNIX **mail** command is probably no big deal. If, however, your mailing list is patently enormous, this method may pose a problem.

One solution is to wrap your **pgp** and **mail** command calls within a **csh** or **tsh** shell script. Then, you simply input the name of the scriptfile to encrypt and mail your message. Be sure to set the script's **chmod** permissions to owner read, write, and executable. Such a scriptfile might resemble [Listing A](#).

Here, line four of the script is a call to **pgp** with options for encryption, ASCII armor, and signature attachment. The file to be encrypted is `file.t`, and the multiple recipients for the encrypted note are `s@cs.ny` and `w@cs.ny`. The encrypted file will be given the default name,

`file.t.asc`. Line six of the script is a loop that mails the encrypted file to each member in the set `R`.

Please note that *command-line calls are not carried over to subsequent lines*. Page-layout constraints in [Listing A](#) make line 5 appear to carry over to the next line. The actual file has the text **foreach R (s@cs.nyu.edu w@cs.nyu.edu) all on the same line**. [Listing B](#) presents a sample

Listing A: A *csh* script for encrypting and mailing a file to multiple parties

```
#!/bin/csh
unset noclobber

pgp -eas file.t s@cs.ny w@cs.ny

foreach R (s@cs.nyu.edu w@cs.nyu.edu)
    mail $R < file.t.asc
end
set noclobber
```

Listing B: An encrypted PGP message sent to multiple parties using a */csh* script, then retrieved and decrypted

```
{s}51: more file.t

Gabba Gabba Hey!
This is only a test.

{s}52: more runscript

#!/bin/csh
unset noclobber

pgp -eas file.t schwarz@cs.nyu.edu m7@ny

foreach R (schwarz@cs.nyu.edu m7@ny)
    mail $R < file.t.asc
end
set noclobber

{s}53: chmod 700 runscript

{s}54: runscript

Pretty Good Privacy(tm) 2.6.2 - Public-key encryption
for the masses.(c) 1990-1994 Philip Zimmermann,
Phil's Pretty Good Software. 11 Oct 94 Uses the
RSAREF(tm) Toolkit, which is copyright RSA Data
Security, Inc. Distributed by the Massachusetts
Institute of Technology. Export of this software may
be restricted by the U.S. government.
Current time: 1998/04/20 19:50 GMT

A secret key is required to make a signature. You
specified no user ID to select your secret key, so
```

the default user ID and key will be the most recently added key on your secret keyring.

You need a pass phrase to unlock your RSA secret key. Key for user ID "Marc Schwarz <schwarz@cs.nyu.edu>"

Enter pass phrase: <Masked>

Pass phrase is good.

Key for user ID: Marc Schwarz <schwarz@cs.nyu.edu> 1024-bit key, Key ID B1953FED, created 1998/03/20 Just a moment....

Recipients' public key(s) will be used to encrypt. Key for user ID: Marc Schwarz <schwarz@cs.nyu.edu> 1024-bit key, Key ID B1953FED, created 1998/03/20

Key for user ID: m7@ny 1024-bit key, Key ID 0007B289, created 1998/04/20

WARNING: Because this public key is not certified with a trusted signature, it is not known with high confidence that this public key actually belongs to: "m7@ny". But you previously approved using this public key anyway.

Transport armor file: file.txt.asc

{s}55: cd

{s}56: mail

mailx version 5.0 Thu May 2 21:00:21 PDT 1996
Type ? for help.

"/mailx/schwarz": 1 message 1 unread
>U 1 schwarz@cs Mon Apr 20 15:50 33/1398

? set crt
? 1

Message 1:
From schwarz@cs.nyu.edu Mon Apr 20 15:50 EDT 1998
Date: Mon, 20 Apr 1998 15:50:56 -0400
From: schwarz@cs.nyu.edu (Marc Schwarz)
To: schwarz@cs.nyu.edu

-----BEGIN PGP MESSAGE-----

Version: 2.6.2
hIwD8LRj5bGVP+0BA/4t9BRwVInLA8nYm7SRxz5IG7G4GNURMAcPn7r
VeFW52uQERKueat3XvBtMYFXEAL05SqFaVlQEQgRLImn9cYcR
2M2Zs0kuqkBzYJd0RUqakgZ60FALtIsbenh9FrI7gdWQxz4XFP1
/mzuY14xhWGs/Sx3pMSxpn8Lr15eJ4FJUISMA+jwMesAB7KJAQQAzx
NRu0t/wpixIFSl50FTDen2o4QmqMeKkWHxAcLUvziZ6LrNvX
h5lGRYoe6f/W6I0WQy5MdxTh3olopcPGw1X5PqyDRrMx24B+nwt
g3NvwBaG46eQqSo/+LwRpEdY1KFMD5FL/bFyDojfThP4vtS
zyjZUhfzahWkrM9caEoeY3qmAAAA3J1aLlI0XtC1CBwEq3Slj9IQK/
ljsYFv4nIiKXN7jSa5dsxkiLy07RWIUWUrKjamOhRbviIv9/ji7c
G/mXgpToYJBTUR0wQhimr60vWSHbkvpy6mhwJkuWrQKeQBquJ0
yqqAIHphoqtSE2/8xxAfaNbXYyoiUoMj5f32ASneYSmLQjqDmsbUN14/
KD7vv8de6Wq98WZX+fkq6Ucjbaiq3LWLbzzf6Vprsl2h/n4H2/
dL21BBdGbpN3EUNPKWHwbvR6D3blw/Ez0VimUkh+hFsiwSf/
nfn0n3mDB0b28==nT1Q

-----END PGP MESSAGE-----

? s 1 MESSAGE.asc
"MESSAGE.asc" [New file] 33/1398
? q

{s}57: pgp MESSAGE.asc

Pretty Good Privacy(tm) 2.6.2 - Public-key encryption for the masses.(c) 1990-1994 Philip Zimmermann, Phil's Pretty Good Software. 11 Oct 94 Uses the RSAREF(tm) Toolkit, which is copyright RSA Data Security, Inc. Distributed by the Massachusetts Institute of Technology. Export of this software may be restricted by the U.S. government.
Current time: 1998/04/20 20:04 GMT

File is encrypted. Secret key is required to read it.
Key for user ID: Marc Schwarz <schwarz@cs.nyu.edu>
1024-bit key, Key ID B1953FED, created 1998/03/20

You need a pass phrase to unlock your RSA secret key. Enter pass phrase: <Masked>
Pass phrase is good. Just a moment.....

File has signature. Public key is required to check signature.

Good signature from user "Marc Schwarz <schwarz@cs.nyu.edu>".
Signature made 1998/04/20 19:51 GMT

Plaintext filename: MESSAGE

{s}58: more MESSAGE

Gabba Gabba Hey!
This is only a test.

execution of such a script and includes the command-line inputs for the retrieval and decryption of the sent message. Actual user inputs are displayed in boldface.

Archiving with PGP

For security reasons, you must frequently control the storage and distribution of information within your organization, realizing that corporate espionage is a very real, and ever-present, danger. Mission-critical documents, such as Ghant charts and R&D memoranda, are prime targets for theft or tampering. Fortunately, PGP can do more for you than just protect your E-mail.

Employee records, while seemingly not as glamorous as trade secrets, also need to be protected, since they're subject to right-to-privacy laws at both state and federal levels. If your facility stores such files online in an unsecured fashion and those private records become compromised, then your firm may be subject to

class-action litigation. In such cases, protecting the employees translates to protecting the company.

It's also important to realize that not all security breaches are external, or even malicious. Honest mistakes happen every day in the business world. The premature distribution of a press release that announces a new, and non-shipping, product can ruin a company. The Osborne Computer Corporation, for example, went into bankruptcy after such an announcement led to massive cancellations of orders for its then current product line. Using PGP encryption to archive and quarantine such promotional documents can help guard against their accidental or inadvertent disclosure.

You archive documents in PGP via conventional cryptography. *Conventional cryptography* differs slightly from standard PGP encryption. A primary benefit of conventional cryptography is that public and private key sets are not used.

Listing C: Encryption and decryption via conventional cryptography

```
{s}87: more file.t

Gabba Gabba Hey!
This is only a test.

{s}88: pgp -ca file.t

Pretty Good Privacy(tm) 2.6.2 - Public-key
encryption for the masses.(c) 1990-1994 Philip
Zimmermann, Phil's Pretty Good Software. 11 Oct
94 Uses the RSAREF(tm) Toolkit, which is
copyright RSA Data Security, Inc. Distributed by
the Massachusetts Institute of Technology. Export
of this software may be restricted by the U.S.
government.
Current time: 1998/04/21 01:11 GMT

You need a pass phrase to encrypt the file.
Enter pass phrase: <Masked>
Enter same pass phrase again: <Masked>
Just a moment....

Transport armor file: file.t.asc

{s}89: more file.t.asc

-----BEGIN PGP MESSAGE-----
Version: 2.6.2

pgAAADnzV/Xbo/p+tmrMXqWAqB2vp7zxr254a62p+ux5N
Dwc5J05RoYB6STZWQevRvpuu0t6zUg4PUOGUqw==o8Xp

-----END PGP MESSAGE-----

{s}90: mv file.t.asc file.OLD.asc
{s}91: pgp -ca file.t

Pretty Good Privacy(tm) 2.6.2 - Public-key
encryption for the masses.(c) 1990-1994 Philip
Zimmermann, Phil's Pretty Good Software. 11 Oct
94 Uses the RSAREF(tm) Toolkit, which is
copyright RSA Data Security, Inc. Distributed by
the Massachusetts Institute of Technology. Export
of this software may be restricted by the U.S.
government.
Current time: 1998/04/21 01:13 GMT

You need a pass phrase to encrypt the file.
Enter pass phrase: <Masked>
Enter same pass phrase again: <Masked>
Just a moment....

Transport armor file: file.t.asc

{s}92: more file.t.asc

-----BEGIN PGP MESSAGE-----

Version: 2.6.2

pgAAADmPweW3VDSY5Vym699BafVne/Vqeg8oPGOXwq0y+
JNEdFYQf8MJRyo2lIu+T5RgFxFxR90VuPbtPS1Ec==WPIE

-----END PGP MESSAGE-----

{s}93: mail m7@ny < file.t.asc
<User m7 then retrieves and decrypts the message>

ny> whoami
m7

ny> more file.t.asc

Date: Mon, 20 Apr 1998 21:23:48 -0400
From: schwarz@granny.cs.nyu.edu (Marc Schwarz)
Message-Id:
<199804210123.VAA26695@granny.cs.nyu.edu>
To: m7@ny
Status: 0
X-Status:

-----BEGIN PGP MESSAGE-----
Version: 2.6.2

pgAAADmPweW3VDSY5Vym699BafVne/Vqeg8oPGOXwq0y+
JNEdFYQf8MJRyo2lIu+T5RgFxFxR90VuPbtPS1Ec==WPIE

-----END PGP MESSAGE-----

ny> pgp file.t.asc

No configuration file found.
Pretty Good Privacy(tm) 2.6.2 - Public-key
encryption for the masses.(c) 1990-1994 Philip
Zimmermann, Phil's Pretty Good Software. 11 Oct
94 Uses the RSAREF(tm) Toolkit, which is
copyright RSA Data Security, Inc. Distributed by
the Massachusetts Institute of Technology.
Export of this software may be restricted by the
U.S. government.
Current time: 1998/04/21 01:32 GMT

File is conventionally encrypted.
You need a pass phrase to decrypt this file.
Enter pass phrase: <Masked>
Just a moment....

Pass phrase appears good. .
Plaintext filename: file.t

ny> more file.t

Gabba Gabba Hey!
This is only a test.
```


Instead, the pass phrase is independent; it can be distributed among others on a need-to-know basis. If your archivist is fired or transferred to the Aleutians, you can still access the archives without having to track him down and beg him for his private key and personal pass phrase.

The following PGP command sequence is for performing conventional cryptography:

```
pgp -ca (filename)
```

Under no circumstances should you use your own personal pass phrase when encrypting with conventional cryptography. Doing so may compromise the safety of your private key, since anyone with root access will then be able to obtain your public and private key set.

Listing C on page 11 shows a session in which files are encrypted and retrieved with conventional cryptography. Note that even when you use conventional cryptography to encrypt the same file twice with identical pass phrases, the resulting encrypted files are slightly different.

Controlling access and distribution of hypersensitive data

If you're in charge of a secured network system, you'll often require very precise levels of data access and control. For example, you may regularly send out confidential office memoranda that you want your recipients to view once and discard.

Listing D: Sample degaussing sessions

```
{s}57: pgp -ea file.t m7
```

```
Pretty Good Privacy(tm) 2.6.2 - Public-key encryption for
the masses. (c) 1990-1994 Philip Zimmermann, Phil's
Pretty Good Software. 11 Oct 94 Uses the RSAREF(tm)
Toolkit, which is copyright RSA Data Security, Inc.
Distributed by the Massachusetts Institute of Technology.
Export of this software may be restricted by the U.S. government.
Current time: 1998/04/21 00:46 GMT
```

```
Recipients' public key(s) will be used to encrypt. Key
for user ID: M7@ny 1024-bit key, Key ID B1953FED, created
1998/03/20.
```

```
Transport armor file: file.t.asc
```

```
<message is then emailed to m7>
```

```
ny> whoami
m7
```

```
ny> pgp -w file.t.asc
```

```
Pretty Good Privacy(tm) 2.6.2 - Public-key encryption for
the masses. (c) 1990-1994 Philip Zimmermann, Phil's
Pretty Good Software. 11 Oct 94 Uses the RSAREF(tm)
Toolkit, which is copyright RSA Data Security, Inc.
Distributed by the Massachusetts Institute of Technology.
Export of this software may be restricted by the U.S.
government.
Current time: 1998/04/21 10:46 GMT
```

```
File file.t.asc wiped and deleted.
```

```
<OOPS !!! File was wiped before it was viewed. Later
that day...>
```

```
ny> pgp file.t.asc
```

```
Pretty Good Privacy(tm) 2.6.2 - Public-key encryption
for the masses. (c) 1990-1994 Philip Zimmermann,
Phil's Pretty Good Software. 11 Oct 94 Uses the
RSAREF(tm) Toolkit, which is copyright RSA Data
Security, Inc. Distributed by the Massachusetts
Institute of Technology. Export of this software may
be restricted by the U.S. government.
Current time: 1998/04/21 23:54 GMT
```

```
File is encrypted. Secret key is required to read it.
```

```
Key for user ID: m7@ny 1024-bit key, Key ID 0007B289,
created 1998/04/20
```

```
You need a pass phrase to unlock your RSA secret key.
Enter pass phrase: <Masked>
```

```
Pass phrase is good. Just a moment.....
Plaintext filename: file.t
```

```
ny> more file.t
```

```
Gabba Gabba Hey!
This is only a test.
```

```
ny> pgp -w file.t
```

```
Pretty Good Privacy(tm) 2.6.2 - Public-key encryption
for the masses. (c) 1990-1994 Philip Zimmermann,
Phil's Pretty Good Software. 11 Oct 94 Uses the
RSAREF(tm) Toolkit, which is copyright RSA Data
Security, Inc. Distributed by the Massachusetts
Institute of Technology. Export of this software may
be restricted by the U.S. government.
Current time: 1998/04/22 23:56 GMT
```

```
File file.t wiped and deleted.
```


You may also need to guarantee that sensitive files, once erased, remain permanently irretrievable. In these instances and others, PGP's built-in features provide solutions where greater encryption flexibility is needed.

Degaussing files

You can have PGP immediately erase a file after you've finished with it by typing

```
pgp -w (filename)
```

When you run PGP with the **-w** option, the designated file is removed from your hard drive.

Degaussing in PGP is stronger than the **rm** command in UNIX. With **rm**, a file's corresponding bits are freed. In PGP, the **-w** switch causes

the system to write over the corresponding bits with zeros *before* the bits are freed. PGP **-w** is a very secure way to erase old files, but it's also permanently destructive—and very unforgiving as to typing errors.

Under no circumstances should you use the **-w** switch to either encrypt or decrypt a message until you have first viewed it or made a backup. The **-w** takes option precedence during PGP execution. Typing

```
pgp -wea (file)
```

or

```
pgp -w (file.asc)
```

won't encrypt or decrypt a file and *then* erase it.

Listing E: Encrypting a message view-only to mail to recipient, who can retrieve and decrypt the note

```
{s}49: more f2.t
```

```
Run For Your Lives !  
The Noodles Are Attacking !!!
```

```
{s}50: pgp -eam f2.t m7
```

```
Pretty Good Privacy(tm) 2.6.2 - Public-key  
encryption for the masses. (c) 1990-1994 Philip  
Zimmermann, Phil's Pretty Good Software. 11 Oct 94  
Uses the RSAREF(tm) Toolkit, which is copyright RSA  
Data Security, Inc. Distributed by the Massachusetts  
Institute of Technology. Export of this software may  
be restricted by the U.S. government.  
Current time: 1998/04/21 00:27 GMT
```

```
Recipients' public key(s) will be used to encrypt.  
Key for user ID: m7@ny 1024-bit key, Key ID  
0007B289, created 1998/04/20
```

```
WARNING: Because this public key is not certified  
with a trusted signature, it is not known with high  
confidence that this public key actually belongs to:  
"m7@ny". But you previously approved using this  
public key anyway..
```

```
Transport armor file: f2.t.asc
```

```
{s}51: mail m7@ny < f2.t.asc  
{s}52: whoami  
schwarz
```

```
<Message is delivered to m7, who retrieves and  
decrypts it>
```

```
ny> whoami
```

```
m7
```

```
ny> pgp f2.t.asc
```

```
No configuration file found.  
Pretty Good Privacy(tm) 2.6.2 - Public-key encryption  
for the masses. (c) 1990-1994 Philip Zimmermann,  
Phil's Pretty Good Software. 11 Oct 94 Uses the  
RSAREF(tm) Toolkit, which is copyright RSA Data  
Security, Inc. Distributed by the Massachusetts  
Institute of Technology. Export of this software may  
be restricted by the U.S. government. Current time:  
1998/04/21 00:35 GMT
```

```
File is encrypted. Secret key is required to read  
it.
```

```
Key for user ID: m7@ny 1024-bit key, Key ID 0007B289,  
created 1998/04/20
```

```
You need a pass phrase to unlock your RSA secret key.  
Enter pass phrase: <Masked>  
Pass phrase is good. Just a moment.....
```

```
This message is marked "For your eyes only".  
Display now (Y/n)? y  
Plaintext message follows...
```

```
-----  
Run For Your Lives !  
The Noodles Are Attacking !!!
```

```
Done...hit any key <Space-Bar>
```

```
<SCREEN CLEARS>
```


Instead, such a command *permanently* erases the file, then halts. If you haven't viewed the file, you'll have to obtain another copy. Thankfully, the **-w** option doesn't recognize UNIX wild cards. Therefore, inputting

```
pgp -w *
```

returns an error and halts. **Listing D** on page 12 contains code for a file-degaussing session.

For your eyes only

In some cases, you may want to ensure that your recipient won't archive the messages you send. The **-m** option sets a file to view-only. Type the command-line call

```
pgp -em (filename)
```

to encrypt the file, designated by **(filename)**, so that it won't be saved to disk upon decryption.

It's important to note that the **-m** option is easily defeated with cutting/pasting between windows and screenshot grabbing. The real purpose of the **-m** is to put your recipient on notice that the message contains information that shouldn't be saved, but rather viewed and discarded.

There's a partial workaround if you discover that your view-only messages have been saved to disk on your system. The best course of action is to contact the recipient and request that she degauss the file. If she refuses, or is unavailable, then you must have root access. You also require an established account policy that gives you explicit permission to access confidential (but non-personal) files in individual accounts. If such a policy exists, you can subsequently combine **grep**

with **pgp -w** to identify and degauss any saved files that were originally sent "for your eyes only."

Listing E on page 13 presents a sample session in which a file is encrypted and retrieved with the view-only option.

Rescinding keys

Mistakes can, and do, happen. Key security is crucial to using PGP safely. Keys can become compromised for any number of reasons. When a compromise occurs, it's usually best to rescind your old keys, issue new ones, and promptly notify those with whom you correspond, preferably by telephone. It's also a good idea to follow up with E-mail, sending these people your revoked public key, so they may modify their respective public keyrings. To rescind your public or private key, type this command:

```
pgp -kd (username)
```

You can then degauss your old private key and type **pgp -kg** to issue yourself a new public and private key set. Be sure to revoke your public key before you erase your private key. This step is necessary because you need your private key to rescind your public key.

If you should lose or destroy your private key, you're stuck. The current PGP implementation doesn't yet account for this scenario. PGP's creator, Philip Zimmermann, has stated that a future revision of PGP will address this issue. ❖

Marc Schwarz is a Ph.D. student at the Courant Institute of Mathematical Sciences, New York University. His first computer was an Osborne Model O1.

LOG-FILE MANAGEMENT

"Rotating" files

by Werner Klauser

Since log files tend to grow in size, you must periodically prune them. Rather than merely deleting them however, you might want to keep them for a while. So, what's the proper way to restrain but retain?

/usr/lib/syslog as an example

Solaris 2.6 uses the command script `/usr/lib/newsyslog` to rotate both the `/var/adm/`

messages and `/var/log/syslog` log files. Let's see how it rotates the `syslog` file, as shown in **Listing A**.

Instead of constantly using the names `/var/log` and `syslog`, the first two lines set the environment variables `LOGDIR` and `LOG` to the values `/var/log` and `syslog`, respectively. As a result, you'll need to change only these variable values if you want to use a different directory

Listing A: Using a command script to rotate syslog

```
LOGDIR=/var/log
LOG=syslog
if test -d $LOGDIR
then
  cd $LOGDIR
  if test -s $LOG
  then
    test -f $LOG.6 && mv $LOG.6 $LOG.7
    test -f $LOG.5 && mv $LOG.5 $LOG.6
    test -f $LOG.4 && mv $LOG.4 $LOG.5
    test -f $LOG.3 && mv $LOG.3 $LOG.4
    test -f $LOG.2 && mv $LOG.2 $LOG.3
    test -f $LOG.1 && mv $LOG.1 $LOG.2
    test -f $LOG.0 && mv $LOG.0 $LOG.1
    mv $LOG $LOG.0
    cp /dev/null $LOG
    chmod 644 $LOG
  fi
fi
```

and/or file later. Then, you use **test -d** to see whether the log directory exists before using **cd** to change to the directory.

At this point, you'll also use **test -f** to check whether a particular generation of the log file exists. If so, you move it to a previous generation. If you copy `/dev/null` into the youngest file, it's then shortened to length 0. **mv** doesn't alter the mode; **cp** does. That's why **chmod** is used to set the log file's mode to its proper value. Its entry in the root crontab file as

```
10 3 * * 0 /usr/lib/newsyslog
```

causes the operating system to rotate the syslog files every Sunday morning at 3:10.

You can use this example to rotate your own files. Embedding the code into your crontab file will prune your large files and let you keep several generations. ❖

Werner Klauser is an independent UNIX consultant working near Zurich, Switzerland. While not paragliding, enjoying his girls, or roarin' around on his Harley chopper, he can be reached by E-mail at klauser@klauser.ch or on his Web page at www.klauser.ch.

Beyond the Basics

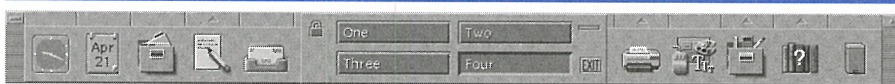
Using CDE

by Don Kuenz

In a joint venture, Sun, IBM, Hewlett-Packard, and Novell developed the Common Desktop Environment (CDE) as an extension to the Motif Graphical User Interface (GUI). Although each must comply with CDE standards, the vendors can add value by implementing new features. Sun adds value by offering an image viewer, which displays images and Postscript.

Sun bundles CDE with Solaris versions 2.5 and later. The company ships it as a free software package that requires its own, separate installation. CDE adds many useful extensions to OpenWindows, which Solaris installs as the default GUI. It also simplifies your interface if you happen to work in a heterogeneous environment that includes IBM or HP workstations.

Figure A



The default CDE front panel allows you to exit and even kill CDE's GUI.

Navigating with keystrokes

If you use a GUI, inevitably there comes a time when your mouse simply stops working. Let's review some common navigation keystrokes *before* your mouse fails. First, press [Alt][Tab] to cycle the input focus between each of the windows on your workspace. CDE includes in the cycle the front panel, as shown in **Figure A**, which provides a way to use keystrokes to exit and kill its GUI.

Use your arrow keys to change the focus within a window. Press [Alt][Spacebar] to make the window's menu appear. After it pops up, you can press m, n, x, or c to either move, minimize, maximize, or close the window, respectively.

SunSoft Technical Support
(800) 786-7638

PERIODICALS MAIL

C:7661905 00002096 04/99

CLINTON TOWNSHIP MI 48035-4218



Please include account number from label with any correspondence.

Figure B



We've modified our CDE front panel.

Customizing the front panel

You can customize CDE's front panel to suit your particular needs. **Figure B** shows a modified front panel, which features five workspace switches with customized names and an **xterm** icon installed in the main panel in place of the default Text Editor (**dtpad**) icon.

You rename a workspace switch by moving the cursor over the switch, right-clicking the mouse button, and selecting Rename from the pop-up menu that appears. You then add a new workspace switch by moving the cursor over any empty, non-icon area of the front panel, right-clicking the mouse button, and selecting Add Workspace from the pop-up menu.

When we click on the small arrow that appears above the **xterm** icon shown in **Figure B**,

the subpanel shown in **Figure C** appears. You can tear off this subpanel and treat it as a separate window. To replace a front panel icon under the subpanel, right-click the mouse over a subpanel icon, and select Copy To Main Panel from the pop-up menu that appears.

Creating a custom action

The **xterm** icon shown in **Figure C** is a custom action. To create a custom action, launch the **dcreate** application. When the window shown in **Figure D** appears, fill in the Action Name. This text appears next to the icon on the subpanel. Finally, enter the command you want to run when this action is pressed.

An action specifies three separate icons. CDE displays one of the three icons based upon available area within a context. You must store the icons in **xpm** format and size them at 16 x 16, 32 x 32, and 48 x 48 pixels. By default, CDE looks for your icons in `$HOME/.dt/icons`. The icon shown with our **xterm** action originally came from a freeware X11 archive; we used an application named **dticon** to convert it to the proper sizes and format.

Summary

Sun bundles CDE as a free add-on to Solaris. It extends Solaris's default OpenWindows GUI with a modifiable front panel that supports multiple workspaces. Using CDE, you can create custom actions to launch your applications. ❖

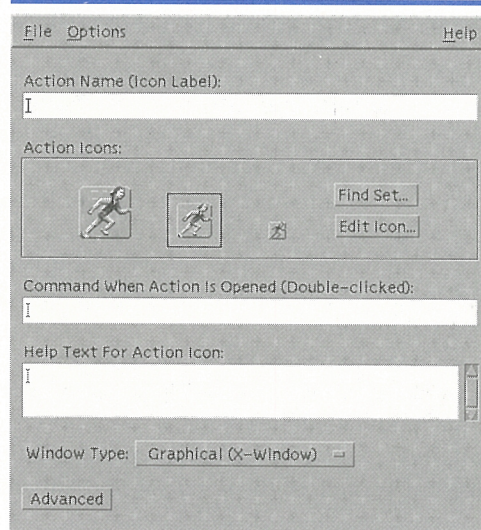
Don Kuenz is a software consultant. You can contact him at <http://gtcs.com/assoclks/don>.

Figure C



You can treat the modified CDE subpanel as a separate window.

Figure D



You can create a new action with **dcreate**'s window

