Volume 6, Number 12 • December 2000
www.elementkjournals.com/sun
U.S. $11.00

Inside Solaris™
Tips & Techniques for users of Sun Solaris

Supercharging third-party libraries with memory-mapped files

by Abdur Chowdhur

DOWNLOAD ftp.elementkjournals.com/sun/dec00

Efficiency in terms of time to market and speed of processing is the name of the game today. Developers are constantly required to develop new products faster and more efficiently. To facilitate that goal, many developers have come to rely on third-party libraries. These libraries allow developers to leverage months and even years of development from specialized libraries and incorporate those features into their products. While this allows products to be developed faster, those products don't always run as efficiently as they would have if the developers had been able to customize the interfaces and features to their application.

In this article, we'll show you how to speed up third-party libraries with file interfaces by using memory-mapped files using the architecture shown in **Figure A**. For more information, see the Solaris mmap man pages. A *file-interfaced library* is a library that takes a filename from the application, opens the file, does some processing, and returns a result.

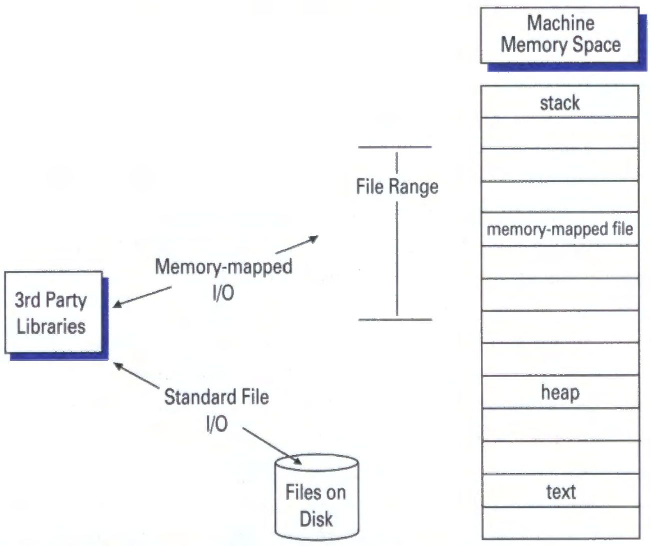

Figure A: *Using a memory-mapped file architecture speeds up third-party libraries.*

A potential performance problem

The problem with file-interfaced libraries is that data must be written from the application's memory buffer to a file, and the library must then open the file and read the data into memory again for processing. If this is done only once, then the overhead may not adversely affect the application's performance. However, if the file processing is done repetitively, then performance can be dramatically hurt.

A better alternative is to pass a memory buffer to the library and have the processing

element k JOURNALS
A Division of Element K Press

done directly in the memory without the overhead of copying the data multiple times and involving the file system. Since many of these libraries don't provide a memory/buffer-passing interface, this option may not be viable.

Another option would be to modify the libraries, but availability of source code or maintenance issues provide additional problems to that solution. Therefore, we need a solution that doesn't require modifying our third-party libraries and still overcomes the performance problems.

Before showing our solution we'll explain the concept of memory-mapped files and how that paradigm can be used for this type of problem. Then we'll present some example code showing both approaches and timing results.

How does memory mapping work?

Memory-mapped I/O maps a file from the file system to the memory system. This allows you to directly manipulate a buffer rather than using read and write calls to access or modify data. So, rather than read/write bytes, you can obtain access via buff[x]=12. Memory-mapping files have been used for many years with virtual memory systems. BSD UNIX has had various flavors of mmap since 1981, and now it's supported by both SVR4- and BSD-based systems (see Gingell, R. A., Moran, J. P., Sannon, W. A. "Virtual Memory Architecture in SunOS," Proceedings of the 1987 Summer USENIX Conference).

The high-level concept is a file that's just a list of blocks of data. By mapping those blocks to the virtual memory system, access from your application is just like any other memory access. In reality, when you attempt to access to that block, a page fault occurs, which tells the virtual memory system to load the page into memory and redirect any access to the new physical memory location. The same process is used for mapping executables files and virtual memory.

File interface solution

While mapping a file to memory may be a good idea for the operating system, when reading data from files, it still requires your third-party libraries to be modified. We've determined this isn't possible.

To circumvent modifying our libraries, we'll map our buffers to files. We'll lock those files into physical memory, and the library will be passed a filename to process. When the library attempts to open and read the file from the file system, those reads will be redirected to physical pages of memory. At least one copy process is eliminated, and file system reads are eliminated and replaced by memory copies. For our application to do this, we must first modify our code.

A typical application allocates a buffer of memory with a call to `malloc`. The system, in turn, allocates memory and returns a pointer to the new memory. The application modifies that buffer with the data for the library. That data is then written out to a temporary file, and the library is called to process it.

While this works fine for buffer interfaces, it becomes a problem when dealing with file interfaces. Another approach is to create a temporary file and seek a given number of bytes into the file (the amount that would have been allocated with `malloc`). Then, write one byte. That new temporary file is then memory mapped. The `mmap` call returns a memory pointer just as `malloc` would have.

Note that new memory can be locked into physical memory via the `mlock` call if you have super user privileges. However, we don't recommend this, since many adverse performance issues are caused by locking memory and removing the paging system's ability to manage system resources. The returned memory buffer is filled with data, just as before.

The library is now sent the name of the temporary file to process. When the library attempts to read from the file, those reads are redirected to physical memory, and only memory copies are required. When the library returns, the temporary file is simply deleted.

A simple mmap example

In our example, we'll show two different applications. The first uses the file system to pass data to an example third-party library. The second does the same function, but utilizes a memory-mapped file.

Using the file system to pass data

Listing A is an example program that uses the file system to pass data to an example third-party library. The program reads in a number of times to run the test and calls `doWork`. Next, `doWork` allocates a memory buffer with `malloc`. Random data is then written to the buffer.

To pass this data to our library, we need a temporary file, so we create one with `fopen`. The data is written to the file with `write`, the temporary memory is freed, and the library is called with the temporary filename. Finally, the temporary file is deleted. This is the typical order of operations for most applications that are using third-party libraries that only support the file interface.

To implement our approach, it needs to be slightly modified. First, rather than using `malloc` to

Listing A: *Using the file system approach*

```
#define TMPDIRFILE "/tmp/nomap.dat"
#define TMPFILESIZE 647680

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>

#include "readfile.h"

int doWork ()
{
   /*
   * Allocate a memory buffer.
   * Write data in the buffer.
   * Write data out to a file.
   * Call readFile function.
   * Delete temp file...
   */
   int     x ;
   FILE    *tmp_fptr ;
   int     ret_stat = 0 ;
   char    *mem_buffer ;

   if((mem_buffer = (char*) malloc(TMPFILESIZE)) == NULL)
   {
      fprintf(stderr,"Error malloc: %d\n", errno ) ;
      return(-1);
   }

   /* *********************************/
   /* Fill the buffer with dummy data */
   for (x = 0; x < TMPFILESIZE; x++)
   {
      mem_buffer[x] = (char) (x&&255) ;
   }

   /* ******************************** */
   /* Open the temp file... */
   tmp_fptr = fopen(TMPDIRFILE, "w");
   if (tmp_fptr == NULL)
   {
      fprintf(stderr,"Can't open tmp file ==> %s: %d\n",
      ➥TMPDIRFILE, errno ) ;
      return(-1);
   }
   /* *********************************/
   /* Write out data...*/
   ret_stat = write (ftell(tmp_fptr), mem_buffer, TMPFILESIZE) ;
   if((ret_stat != TMPFILESIZE) || (ret_stat<0))
   {
      fprintf(stderr,"Error writing data==> %s: %d\n",
      ➥TMPDIRFILE, errno ) ;
      return(-1);
   }
   else
   {
      ret_stat = 0 ;
   }

   /* Free memory */
   free (mem_buffer) ;

   /* Close temp file... */
   fclose(tmp_fptr) ;

   /* Read the temp file.. */
   readFile ( TMPDIRFILE ) ;

   /* Delete the temp file... */
   unlink(TMPDIRFILE);

   return (ret_stat) ;
}

int main (int argc, char **argv)
{
   int     ret_stat = 0 ;
   int     x = 0 ;

   for (x = 0; x < atoi(argv[1]); x++)
   {
      ret_stat = doWork () ;
      if (ret_stat != 0)
      {
         fprintf(stderr,"Error in doWord.\n" ) ;
         break ;
      }
   }

   return (ret_stat) ;
}
```

create memory that will be written out to a temporary file, we use the `mmap` call to allocate a temporary memory-mapped file the same size that our `malloc` would have returned. By removing the `malloc`, we remove at least one level of copy operations, since the temporary memory doesn't need to be copied to the file system.

Our example using memory mapped files

Our `mmap` example program shown in Listing B on the next page is almost identical to our first program. We first read in the number of iterations to run the program. The `doWork` method is called. This time, instead of using `malloc` to create a temporary memory buffer, a temporary file is created with `fopen`. This program uses `lseek` to set the number of bytes the file will contain. This can be considered equivalent to our `malloc` operation, but using the virtual memory manager to manage our process memory space. Next, the random data is written to the memory-mapped file, in the same manner that any memory object would be modified. The temporary file is closed and the `readFile` library method is called, just as in the previous example.

Listing B: *The memory-mapped file approach*

```
#define TMPDIRFILE "/tmp/nomap.dat"
#define TMPFILESIZE 647680

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>

#include "readfile.h"

int doWork ()
{
   /*
   * Allocate a memory buffer.
   * Write data in the buffer.
   * Create dummy file.
   * Mmap data to file.
   * Lock memory.
   * Write data out to a file.
   * Call readFile function.
   * Delete temp file...
   */
   int     fd ;
   int     x ;
   int     ret_stat = 0 ;

   /* Create a memory ptr */
   char    *mem_buffer ;

   /* **********************************/
   /* Open the temp file...           */
   if ((fd = open(TMPDIRFILE,
   ➥O_RDWR|O_CREAT|O_TRUNC, S_IRUSR|S_IWUSR)) < 0)
   {
      fprintf(stderr,"Can't open tmp file ==>
      ➥%s: %d\n", TMPDIRFILE, errno ) ;
      return(-1);
   }

   /* ********************************************/
   /* lseek the temp file to the desired size.  */
   if (lseek ( fd, TMPFILESIZE-1, SEEK_SET ) < 0)
   {
      fprintf(stderr, "Can't seek to %d:
      ➥%d\n", TMPFILESIZE, errno ) ;
      return(-1);
   }

   /* ************************************************/
   /* write one byte, to set the current file size */
   if (write(fd, "", 1) !=1 )
   {
      perror("Write error:");
      return(-1);
   }

   /* **********************************/
   /* Memory map file...              */
   if ((mem_buffer = mmap((caddr_t)0,
   ➥TMPFILESIZE, PROT_READ|PROT_WRITE,
      MAP_SHARED|MAP_NORESERVE, fd, 0))
      ➥== (caddr_t)(-1))
   {
      perror("Error memory mapping file:");
      exit(-1);
   }

   /* **********************************/
   /* Lock the memory,                */
   mlock(mem_buffer, TMPFILESIZE) ;

   /* **********************************/
   /* Fill the buffer with dummy data */
   for (x = 0; x < TMPFILESIZE; x++)
   {
      mem_buffer[x] = (char) (x&&255) ;
   }

   /* Close temp file... */
   close(fd) ;

   /* Read the temp file.. */
   readFile ( TMPDIRFILE ) ;

   /* Unmap the data from the file association*/
   munlock(mem_buffer, TMPFILESIZE) ;
   munmap(mem_buffer, TMPFILESIZE) ;

   /* Delete the temp file... */
   unlink(TMPDIRFILE);

   return (ret_stat) ;
}

int main (int argc, char **argv)
{
   int     ret_stat = 0 ;
   int     x = 0 ;

   /* Preform the test multiple times... */
   for (x = 0; x < atoi(argv[1]); x++)
   {
      ret_stat = doWork () ;
      if (ret_stat != 0)
      {
         fprintf(stderr, "Error in doWork: %d\n", x ) ;

         break ;
      }
   }

   return (ret_stat) ;
}
```

Listing C: *Sample method to read a file*

```
/*
   * Get a file name, open file,
   * allocate memory for file
   * and read it to buffer.
   *
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include "readfile.h"

int readFile ( char *fname )
{
   FILE    *fptr ;
   struct stat stat_buf ;
   int     size_in_bytes ;
   int     bytes_read ;
   char    *mem_buffer = NULL ;
   char    *mem_ptr = NULL ;

   /* Open the file for reading... */
   fptr = fopen(fname, "r");
   if (fptr == NULL)
   {
      fprintf(stderr,"Can't open tmp file
      ➥for reading==> %s: %d\n", fname, errno ) ;
      return(errno);
   }
   /* Get file status... */
   fstat(fileno(fptr), &stat_buf) ;
   size_in_bytes = stat_buf.st_size ;

   /* Get a buffer... */
   mem_buffer = (char*) malloc(size_in_bytes+1) ;
   mem_ptr = mem_buffer ;

   /* Read data... */
   for (;;)
   {
      bytes_read = read (fileno(fptr), mem_ptr, size_in_bytes) ;
      if (((bytes_read<0)) && (errno==EINTR)) continue ;
      if ((bytes_read<0))
      {
         fprintf(stderr,"Error reading:%d\n", errno) ;
         break ;
      }
      if (bytes_read == size_in_bytes) break ;
      if (bytes_read == 0) break ;
      if (bytes_read < size_in_bytes)
      {
         mem_ptr += bytes_read ;
         size_in_bytes -= bytes_read ;
      }
   }
   /* Free memory... */
   free (mem_buffer) ;
   /* Close temp file... */
   fclose(fptr) ;
   return (0) ;
}
```

After the file has been read, the memory is unmapped and the temporary file is deleted.

Running each example

Listing C shows the sample program we used to run each of our examples. While the modification differences of the two programs aren't large, there's a paradigm shift in thinking about what's the most appropriate method for memory management depending on the situation.

Substantial improvement

We ran each application through tests to check performance. Those programs are run using a test data set of 640 KB. The timing results are shown in **Table A**. As you can see, there's a substantial improvement in speed when using the memory-mapped file approach. While your application and situation may differ, having several implementation approaches helps find alternative solutions to bottleneck problems. By creating a temporary file and memory mapping it to physical memory, we can use third-party libraries without the performance penalties that come from normal file-based interfaces. ✳

Table A: *Performance comparison between our two example programs*

Memory mapped	File system
Real	0.063s2.690s
User	0.050s0.030s
Sys	0.010s0.070s

Accessing a DOS formatted disk in Solaris

Unlike a CD-ROM, a disk drive isn't polled automatically because the extra work would quickly ruin it. Using volume management, type `volcheck`. This forces Solaris to poll the drive and you should see something like this after typing `mount`:

```
/floppy/unnamed_floppy on /vol/dev/diskette0/unnamed_floppy read/write on Sun Nov 12 11:32:37 2000
```

To safely eject the floppy, you should unmount the drive by using `eject`. You will get a confirmation message, and then it should be safe to remove the disk.

DOWNLOAD ftp.elementkjournals.com/sun/dec00

Securing BIND on Solaris

by Rob Thomas

The ubiquitous BIND (Berkeley Internet Name Domain) server is distributed with Solaris and provides name services to countless networks. However, the BIND server isn't without certain vulnerabilities, and is often a choice target for Internet vandals. These vandals utilize BIND vulnerabilities to gain root access to the host or to turn the host into a launching platform for DDOS attacks. An improper or insufficiently robust BIND configuration can also leak information about the hosts and addressing within the intranet. In this article, we'll present a template for deploying a secure BIND configuration, thus mitigating some of the risk of running the BIND server.

Secure the BIND configuration

The following configuration presumes that the host network is connected to the Internet. Further, we assume that you have a working knowledge of BIND version 8. (We won't go into detail about the creation of zone files or general DNS configuration and maintenance.) We'll discuss how to supplement and protect an existing BIND configuration. This is a split-brain NS model with a twist—the two sides of DNS are contained within completely distinct spaces. The files and address space are separate, thus providing a great deal of protection against an attack that targets one side of the DNS topology.

Setup

For our example, our internal network is 7.7.7/24, and our external network is 8.8.8/24. Our name server has an NIC in each subnet, with the internal NIC as 7.7.7.1 and the external NIC as 8.8.8.1. The name server is often also the firewall, so the addressing scheme and dual NIC configuration likely fits many network topologies.

Now, ensure that you have the latest version of the BIND. If you don't, you can obtain it by visiting the Internet Software Consortium BIND site at **www.isc.org/products/BIND/**. The compilation and installation of the BIND is well detailed at this site.

Since two BIND daemons will be running on the host, you must create distinct UIDs and directories. First, create two UIDs and GIDs, intnamed and extnamed. We'll use the intnamed user and group for our internal DNS, and use the extnamed user and group for our external DNS. Now, assign each a unique, non-zero UID and GID. We recommend using a shell that will allow no access, such as /bin/false or our own denial shell, nocando. The nocando denial shell, which includes detailed logging, can be found at **www.enteract.com/~robt/Tools**. The home directories for these accounts will be the directories we create in the next step.

Next, create two directory structures to contain your configuration files. For example, use /var/named/extnamed and /var/named/intnamed. In each directory, create a subdirectory named *master* for your master zones and *slave* for your slave zones, if such zones exist.

Configure the zone files normally, and place them in the proper subdirectories. Remember to place the internal zone files in /var/named/intnamed, and the external zone files in /var/named/extnamed. Each subdirectory must also have a hint file and a named.conf file. Don't forget to create a loopback zone file for each subdirectory. You can find details on how to create hint and zone files in the August 2000 article "Configuring BIND 8" (**www.elementkjournals.com/sun/s_sun/0008/sun0086.htm**) by Don Kuenz.

Let's now configure the two named configuration files, one for internal DNS and another for external DNS. Pay close attention to the subtle differences between them.

Configuring the external DNS—ext-named.conf

The external DNS configuration file example shown in **Listing A** is built to service externally-based queries, recursively answer internally-based queries, and to mitigate many of the attacks commonly found on the Internet. If an attack does succeed in breaching or halting the externally named daemon, service on the internally named daemon for queries of internal names and addresses is largely not effected.

Configuring the internal DNS—int-named.conf

The internal DNS configuration file is built to service queries from hosts on the internal network(s). It's also built to forward queries for external hosts to the named daemon servicing the external side of the name server. Our example is shown in **Listing B** on page 8.

Listing A: *The BIND configuration file for our external DNS*

```
// @(#)ext-named.conf 12 NOV 2000 Rob Thomas robt@cymru.com
// Set up our ACLs
acl "xfer" {
    none; // Allow no transfers. If we have other
            // name servers, place them here.
};

acl "bogon" {
// Filter out the bogon networks. These are networks listed
// by IANA as test: RFC1918, Multicast, experimental, etc.
// If you see DNS queries or updates with a source address
// within these networks, this is likely of malicious origin.
    0.0.0.0/8;
    1.0.0.0/8;
    2.0.0.0/8;
    169.254.0.0/16;
    192.0.2.0/24;
    10.0.0.0/8;
    172.16.0.0/12;
    192.168.0.0/16;
    224.0.0.0/3;
    240.0.0.0/4;
};

// Set options for security
options {
    directory "/var/named/ext-named";
    pid-file "/var/named/ext-named/ext-named.pid";
    statistics-file "/var/named/ext-named/ext-named.stats";
    dump-file "/var/named/ext-named/ext-named.dump";
    query-source address 8.8.8.1;

    listen-on { 8.8.8.1; };
        // Listen on our external interface only.

    allow-transfer {
        // Zone tranfers limited to members of the "xfer" ACL.
            xfer;
    };

    allow-query {
        // Accept all queries except from the bogons.
        any;
        };

    blackhole {
        // Deny anything from the bogon networks as
        // detailed in the "bogon" ACL.
            bogon;
        };
};

// Link in our zones
zone "." in {
    type hint;
    file "db.cache";
};

// Ensure that all attempts to query for BIND.TXT are logged
zone "bind" chaos {
    type master;
    file "master/db.bind";

    allow-query {
    none;
    };

    allow-transfer {
        none;
    };
};

// Allow queries for the 127/8 network, but not zone transfers.
// Every name server, both slave and master, will be a master
// for this zone.
zone "0.0.127.in-addr.arpa" in {
    type master;
    file "master/db.127.0.0";

    allow-transfer {
        none;
    };
};

zone "ournetwork.net" in {
    type master;
    file "master/db.ournetwork";
};

zone "8.8.8.in-addr.arpa" in {
    type master;
    file "master/db.8.8.8";
};
```

Configuring the internal DNS—db.cache

The hint file for the external DNS server should be the standard hint file. However, the hint file for the internal DNS server should be configured with only the names and addresses of your external DNS server(s). An example would be:

```
; @(#)db.cache for internal DNS 12 NOV 2000
;   Rob Thomas robt@cymru.com
.   99999999    IN  NS  ns1.ournetwork.net.

;
; hotwire the addresses
;

ns1.ournetwork.net.      IN  A   8.8.8.1
```

The bind zone file

A common trick of the malicious is to query the name server for the version of BIND code it's running. This allows the attacker to quickly select a version-specific vulnerability. To hide the version number, you could insert the version directive in the options section. However, this doesn't provide any logging of the attempt. Instead, we created a bind zone file, linked it into both our external and internal configuration, and now we receive notification of any attempts to query for version.bind. The code below shows us how:

```
; @(#)db.bind 12 NOV 2000 Rob Thomas robt@cymru.com
$TTL    1D
$ORIGIN bind.
```

Listing B: *The internal DNS configuration file for BIND*

```
// @(#)int-named.conf 12 NOV 2000
// Rob Thomas robt@cymru.com
// Set up our ACLs

acl "xfer" {
        none; // Allow no transfers. If we have other
              // name servers, place them here.
};

acl "trusted" {
   // Place our internal subnet in here so
   // that intranet clients may send DNS queries.
   7.7.7.0/24;
   localhost;
};

acl "bogon" {
   0.0.0.0/8;
   1.0.0.0/8;
   2.0.0.0/8;
   169.254.0.0/16;
   192.0.2.0/24;
   10.0.0.0/8;
   172.16.0.0/12;
   192.168.0.0/16;
   224.0.0.0/3;
   240.0.0.0/4;
};

options {
   directory "/var/named/int-named";
   pid-file "/var/named/int-named/int-named.pid";
   statistics-file "/var/named/int-named/int-named.stats";
   dump-file "/var/named/int-named/int-named.dump";
   listen-on { 7.7.7.1; };

   forwarders { 8.8.8.1; };
   // Send all queries for external hosts to the external
   // named on this same host. The external named daemon
   // will recursively seek the answer to the internal query,
   // then return the answer.

   allow-transfer {
      xfer;
   };

   allow-query {
      trusted;
   };

   blackhole {
      bogon;
   };
};

zone "." in {
   type hint;
   file "db.cache";
};

zone "bind" chaos {
   type master;
   file "master/db.bind";

   allow-query {
      none;
   };

   allow-transfer {
      none;
   };
};

zone "0.0.127.in-addr.arpa" in {
   type master;
   file "master/db.127.0.0";

   allow-transfer {
      none;
   };
};

zone "internal.ournetwork.com" in {
   type master;
   file "master/db.internal";
};

zone "7.7.7.in-addr.arpa" in {
   type master;
   file "master/db.7.7.7";
};
```

```
@   CHAOS   SOA localhost. root.localhost. (
      2000081201  ; serial
      3H        ; refresh
      1H        ; retry
      1W        ; expiry
      1D )          ; minimum
    CHAOS NS     localhost.
```

If anyone attempts to obtain the version number, the logs will report the following:

```
Nov 12 18:28:42 ns1 named[17809]: unapproved
➥query from [X.X.X.X].33112 for "version.bind"
```

Add this bit of text to your alert mechanism!

Starting the two named daemons

Once the configuration is complete, it's time to launch the two named daemons. Without changing /etc/init.d/inetsvc, it's wise to test the configurations first. To start the external DNS service, type the following:

```
/usr/local/sbin/named -u extnamed -c
➥/var/named/extnamed/ext-named.conf
```

By typing the `ps -fu extnamed` command, you can verify that your externally named daemon is running. To start your internal DNS service, type this:

```
/usr/local/sbin/named -u intnamed -c
➥/var/named/intnamed/int-named.conf
```

Again, typing `ps -fu extnamed` will verify that your internal name service is up and running.

Once you have /etc/resolv.conf properly configured (use the internal address for the nameserver line), you should be able to launch queries. Test a few basic queries for both internal and external names and addresses. Then, test an intranet client to ensure that the forwarding is properly working. If you have secondary name servers, ensure that zone transfers are functional and complete.

Once you've successfully completed all the testing, modify /etc/init.d/inetsvc to start two named daemons at boot. Comment out the currently named lines and insert the code in **Listing C**.

Listing C: *Changes made to inetsvc for running our two named daemons*

```
# Start external DNS
if [ -f /usr/local/sbin/named -a -f
➥/var/named/extnamed/ext-named.conf ];
then
    echo "Starting external BIND server."
   /usr/local/sbin/named -u extnamed -c
   ➥/var/named/extnamed/ext-named.conf
fi

# Start internal DNS
if [ -f /usr/local/sbin/named -a -f
➥/var/named/intnamed/int-named.conf ];
then
   echo "Starting internal BIND server."
   /usr/local/sbin/named -u intnamed -c
   ➥/var/named/intnamed/int-named.conf
fi
```

Increased security

BIND is a necessary application on almost all networks. However, it isn't without certain risks. By implementing a secure configuration and a separation of internal and external name services, you can increase the security and reliability of your name services and protected networks. ✳

Auditing Solaris security with CLI

by Boris Loza

Did you know that you can inspect the security on an existing Solaris box by using the command line interface (CLI)? You don't have to install any expensive GUI-based applications. In this article, we'll build a security check list using just native Solaris OS commands. Following this list step by step will help you to identify whether the system fits with your security policy.

In this article we won't go into specifics about the need for particular checks. For a detailed explanation about Solaris security, please refer to *Practical UNIX and Internet Security* by Simson Garfinkel and Gene Spafford. Note that the pound sign (#) in front of a UNIX command indicates that this command should be executed by root.

Gathering background information

The first step is to get information about your system. You do this with the following `uname` command:

```
uname -a
```

Using this output, we can tell the OS version (e.g., SunOS 5.6) the name of the hardware implementation (e.g., sun4m sparc SUNW, Ultra-2) and whether the latest kernel patch has been installed. Getting Generic_105181-22 for Solaris 2.6 SPARC indicates that you have the latest kernel patch (at the time this article was written). You can obtain the latest kernel patch from http://sunsolve.sun.com.

You need to know the OS version and the hardware implementation for applying OS/hardware-specific security patches. To display the patches installed, type the following:

```
showrev -p
```

This prints all patches currently installed on the system. You can compare the output with the list of recommended security patches available for this OS version. The latest recommended security patches are also available from Sunsolve.

Checking account security

The next step is to check your user accounts. First, display accounts without a password:

```
#logins -p
```

Then, delete such accounts immediately or set passwords for them. Now, you check accounts with duplicate UIDs:

```
#logins -d
```

You'll want to provide a different UID for all accounts on your system.

Now you can display the date of the last password change, minimum number of days required between password changes, and maximum number of days the password is valid:

```
#passwd -sa
```

To alter any of these password attributes, edit the /etc/default/passwd file. Now, display inconsistencies in the password file:

```
/usr/sbin/pwck
```

This tells you about accounts with no login directory and the wrong shell. Next, display any inconsistencies in the group file:

```
/usr/sbin/grpck
```

Now you can check the accuracy of file attributes of installed files:

```
/usr/sbin/pkgchk -a
```

You can fix any inconsistencies you find manually. In addition, you may want to run the fix-modes utility found at ftp.fwi.uva.nl/pub/solaris/fix-modes.tar.gz. It will fix all mode 755 directories and binaries and change the ownership to root where needed. Currently it supports Solaris 2.2 to Solaris 8.

Now you can display system parameters with the following:

```
cat /etc/default/login
```

Pay attention to the CONSOLE, PASSREQ and UMASK variables. The CONSOLE and PASSREQ variables must be uncommented. UMASK should be set to 022 or 025.

Next, display the password, the shadow and the group files with the following:

```
cat /etc/passwd
#cat /etc/shadow
cat /etc/group
```

Make sure that /dev/null is the shell for all non-root users in /etc/passwd. See if you have NP – no password entry for all system accounts in /etc/shadow.

Network controls

The network can be a source of many security concerns. You can do some simple checks to help ensure you're properly configured. First, display trusted hosts and users:

```
cat /etc/hosts.equiv
```

No trusted hosts should be allowed. You can delete this file. Next, display NFS files and parameters:

```
cat /etc/dfs/dfstab
```

Consult *Practical UNIX and Internet Security* for how to improve NFS security. Now, display the message of the day file:

```
cat /etc/motd
```

This file should contain a warning to unauthorized users stating that they aren't welcome.

Now, display unauthorized statement at login:

```
cat /etc/issue
```

Do the same to this file that you did for /etc /motd. Next, display the network services file:

```
cat /etc/inetd.conf
```

This file should contain services only used by your system. Comment out any unused services. Display the system accounts that aren't allowed to use FTP to transfer files:

```
cat /etc/ftpusers
```

This file should contain all system accounts including root. Display network services currently active:

```
rpcinfo -p
```

Make sure that it isn't running any processes that aren't needed (e.g., rstatd, rusersd and rexd). Now, display the version of sendmail:

```
# /usr/ccs/bin/what  /usr/lib/sendmail
```

It's always better to have the latest version of sendmail installed on your machine. For information about the most recent sendmail implementation, visit www.sendmail.org.

Now, display the network rhost and netrc files with the following:

```
#find / -name .rhosts -ls
#find / -name .netrc -ls
```

These files don't have to exist on the system. To disable the user's ability to create .rhosts files, edit the /etc/pam.conf file (Solaris 2.6 and higher). If you can't get rid of these files, make sure that their permissions are 600 and a user in whose home directory they are located owns them. Now, display the user's profile file permissions for different shells:

```
#find / -name .profile -ls
#find / -name .login -ls
#find / -name .cshrc -ls
#find / -name .kshrc -ls
```

The user should be an owner of his profile. These files should only be readable by the owner.

Monitoring and logging

Logging can provide a wealth of useful security information if you set it up correctly. We can do this in a few steps. First, display the system events to log:

```
cat /etc/syslog.conf
```

Although this file is installed by default, its configuration should be adjusted to specify what messages are to be stored in what files or forwarded to another loghost on the local network.

By default, Solaris doesn't capture syslog events sent to LOG_AUTH. This information is very useful since it contains information on unsuccessful login attempts, successful and failed su attempts, reboots, and a wealth of other security-related information. Consult www.cert.org/security-improvement/implementations/i041.08.html for detailed syslogd configuration information. Now, display accounts that use the su command:

```
#cat /var/adm/sulog
```

Checking the sulog will tell you if your users are trying to become the root by searching for passwords. If you see dozens of su attempts from a particular user who isn't supposed to have access to the root account, you might want to ask him what he's trying to do.

Display audit events that have been defined:

```
cat /etc/security/audit_control
```

This file contains audit control information used by auditd. Note that the functionality of this file is available only if the Basic Security Module (BSM) has been enabled.

Display the following if the logging cron is enabled:

```
cat /etc/default/cron
```

The CRONLOG variable should be set to YES. Now, check for all failed login attempts with this:

```
cat /var/adm/loginlog
```

After five unsuccessful login attempts, all the attempts are logged in the /var/adm/loginlog file. By default this file doesn't exist, so no logging is done. To enable logging, create the /var/adm /loginlog file. Change permissions to 600. The owner of this file must be root. The group must be set to sys.

File and directory permissions

Files and directories can pose many security issues. Permissions need to be set to reflect your current permission policies to minimize the chances of system damage. First, display file permissions in the root directory:

```
ls -la /
```

Look for any unusual files in the root directory. Now, display file permissions in the /etc directory:

```
ls -la /etc
```

All the files in /etc should be kept unwritable by users other than root. Display file permissions in the /etc/default directory:

```
ls -la /etc/default
```

No files with write permissions are allowed in this directory. Now, display file permissions over system log files:

```
ls -la /var/adm
```

All files in this directory must be owned by system accounts (not actual human accounts) and not have world write permissions. Next, display file permissions for the scheduled files in the root crontab:

```
ls -l /var/spool/cron/crontabs/root
```

This file must have 400 permissions and be owned by root only. Display file permissions for the cron log file:

```
ls -l /var/cron/log
```

This file must have 600 permissions and must be owned by root. Display files owned by non-existent users or groups:

```
# find / \( -nouser -o -nogroup \) -ls
```

Delete these files or change the ownership for existing users and groups. Now, display SUID and SGID files owned by root:

```
# find / -user root \( -perm -4000 -o -perm
➥-2000 \) -ls
```

It's a good idea to run this command soon after the system has been set up. Send the output to a file, and keep it for making comparisons later:

```
# find / -user root \( -perm -4000 -o -perm
➥-2000 \) > files.check
```

Then once in a while run the following command:

```
# find / -user root \( -perm -4000 -o -perm
➥-000 \) | diff - files.check
```

One of the ways to grant users the root privileges to do a specific job is by using the sudo application found at **http://smc.vnet.net/**. Now, display world-writeable files:

```
# find / -type f -perm -2  -ls
```

Most systems don't have any reason to have files that are writeable by anyone. Next, display world-writeable directories:

```
# find / -type d -perm -2 -ls
```

Review the output. You don't need it set for directories such as /etc, /var, /dev, /devices etc.

Other areas

You can find out the last time your system was rebooted with the following command:

```
last reboot /var/adm/wtmp
```

If you have accounts with a restricted shell, check to see if they're set up properly. The vanilla restricted shell rksh is breakable. To check whether restricted shell accounts are set up correctly, do the following:

- From the restricted shell account, start the vi editor.
- Inside of vi, set the following variable:

  ```
  :set shell=/bin/sh
  ```

- Type `:shell`. When you get a shell prompt, try `cd /`. If you succeed, your restricted shell account should be configured properly.

For how to configure a restricted shell account, refer to *Practical UNIX and Internet Security*.

It's always a good idea to check to see which kernel modules are loaded. Programs such as TTY Watcher, which can capture all users' keystrokes need to be loaded into the kernel. Make sure that no foreign modules are loaded. You can check which kernel modules are loaded by typing the following command:

```
modinfo
```

Conclusion

By using the operating system's commands, you can quickly check the security on your Solaris machine. You don't have to install and configure fancy security checking applications.

The security checklist provided in the article doesn't intend to be unique or complete. You can expand it with whatever your specific security needs may be. Consider your environment when looking at security. Your approach to a machine that is available on the Internet will be much different than a box located behind a firewall with a non-routable address. ✳

Using truss to track processes

by Shriman Gurung

If a process tries to use a file and fails, for whatever reason, you can use the truss command to get to the bottom of the problem. Use truss to trace the system calls and file accesses that a process makes. You'll find that truss is ideal for solving problems involving missing files and permissions.

Let's illustrate with an example. On our system, the file /tmp/foo doesn't exist. If we try to list it, we get the following familiar error:

```
$ ls /tmp/foo
/tmp/foo: No such file or directory
```

Now let's peek under the hood with truss. We simply call truss along with the name of the program we want to look at and any arguments that the program normally takes. You'll see that truss returns a lot of information, so for clarity we'll only show the most relevant lines of output:

```
$ truss ls /tmp/foo
execve("/usr/bin/ls",0xFFBEB4,0xFFBEC0) argc=2
.
.
lstat64("/tmp/foo",0xFFBEFA58)     Err#2 ENOENT
.
.
```

Solaris uses the `execve` system call to start the `ls` program. Then it attempts to retrieve information about the /tmp/foo file using the `lstat64` system call. This fails with return code `ENOENT`; that is, the file doesn't exist. `ENOENT` is a standard Solaris error code; for a complete list, see the header file /usr/include/sys/errno.h.

Saving truss' output to a file

Let's try another example. In Solaris, only root may read the /etc/shadow file, since it holds encrypted passwords for all accounts on the system. Let's try reading it as a non-root user:

```
$ truss cat /etc/shadow 2> /tmp/truss.out
$ more /tmp/truss.out
.
.
```

This time we redirect the output of truss to a file to make it easier to read. Truss sends its output to standard error in order not to conflict with the normal behavior of your program, which it assumes will send its results to the standard output.

Looking at the output from truss, we see a line like this:

```
open64("/etc/shadow",O_RDONLY)  Err#13 EACCES
```

What does this mean? Truss is telling us that Solaris tried to open the file, but failed because the calling process didn't have sufficient permission to access the file.

Using truss on a running process

Sometimes it's not appropriate to start a new process with truss. For example, if you wanted to track down a problem with your mail server, you might not have the luxury of being able to shut down the existing process and start it up again with truss.

The `-p` option for truss allows it to monitor a running process. Use `ps` to find the process ID of the program that you're interested in, and then pass this number to truss. For example, let's say we want to monitor sendmail. Sendmail runs as root, so we must run truss as the root user:

```
# ps -e | egrep sendmail
1776 ? 0:00 sendmail
```

The process ID of sendmail is 1776, so we pass that to truss:

```
# truss -p 1776 2> /tmp/truss.out &
# tail -f /tmp/truss.out
```

```
.
.
.
poll(0xFFBEDF70,1,60000)        (sleeping...)
```

This time, we've chosen to run truss in the background by using the suffix &. This allows us to continue working while it collects information. We can also use the tail utility to watch the output file as truss adds to it. Notice how truss reports the process as sleeping when sendmail is waiting for work to do.

Summary

Take a few minutes to play around with truss. Remember that truss is a general-purpose, process-tracking utility. Whenever you want to know what your process is doing, truss is the tool to use. ✳

Low-hassle news pulls

DOWNLOAD ftp.elementkjournals.com/sun/dec00

by Don Kuenz

In the last two months we brought you the articles "Creating a News server" and "Newsbot cleans up." In these articles, we showed you how to configure a full-blown news setup. But what if you just want to automate the pulls of a few newsgroups without all the complications of a dedicated news server? If you'd like to set up some news service capabilities, you can use a free package named *suck* to pull small amounts of news from an ordinary dialup Internet connection into a local news server. You don't need a peering arrangement with your upstream news server, because suck uses the same protocol used by newsreaders. Besides pulling news, you can also post articles to your upstream news server.

In this article, we'll show you how to install and use the package. You should only use it to feed roughly a dozen non-binary news groups. Many news administrators consider it abusive to use this software to pull thousands of groups containing megabytes of data.

Installing the software

You should find it fairly easy to install the software. Suck's authors distribute the package as C source. Note that you must compile it before you can install it. Download a free C compiler and a make tool from **www.sunfreeware.com**. You can obtain suck's source code in an archive file available at **ftp://sunsite.unc.edu/pub/Linux/system/news/transport**.

After you obtain the archive, extract the source and change to the directory that it creates. Next, compile and install suck using the following commands:

```
./configure
gmake
gmake install
```

These commands create binaries that use a traditional, flat-file, news-spool structure. The README distributed with the source explains how to use other spool structures. The default configuration installs the binaries into the /usr/local directory. The README also explains how to install the binaries into a different directory. Depending upon your host's security, you may need to run `gmake install` as root.

The installation procedure installs the binaries shown in **Table A**. It also installs the associated man pages. After you install the binaries and man pages, you can configure your host.

Configuring your host

Configuring your host to pull news involves creating several data files and modifying a script named get.news.inn. You can find samples of both under the samples directory in the source distribution. In our example, we place the data files under a directory named /usr/local/news/db.

At a minimum, you need to create two files named *active-ignore* and *sucknewsrc*. You put the names of local newsgroups that you don't want to pull into the active-ignore file. You can use wildmat characters to specify ranges of groups. Our active-ignore file contains the following lines:

```
control
control.*
junk
local.*
```

The sucknewsrc file contains newsgroup names followed by the last article number pulled. You can initialize this file by adding one line for each newsgroup that you want to pull. The line contains the name of the newsgroup followed by a 1.

Table A: *Binary filenames and their functions*

Binary name	Function
lmove	Takes articles in a single directory and puts them into a directory tree based upon newsgroups
lpost	Reads news data from stdin and posts it to the local server
rpost	Posts local articles to your upstream news server
suck	Pulls a small newsfeed from your upstream news server
testhost	Tests the status (up or down) of a news server

When you pull news for the first time, the software assumes that it needs to pull all articles with an article number greater than 1. Our initial sucknewsrc file contains the following lines:

```
comp.unix.solaris 1
alt.solaris.x86 1
```

You also need to configure a shell script that pulls and posts news articles from your upstream news server. Our script uses at least one argument—the host name of the upstream news server. You can include additional arguments to specify a username and password if your upstream news server requires authentication.

Due to space limitations, we have made our get.news.inn file available for downloading at our FTP site. Let's cover some of the finer points of this sample script. A variable named `REMOTE_HOST` holds all of

About our contributors

Abdur Chowdhury is a senior software engineer at America Online and an adjunct professor at Georgetown University. He's also finishing his Ph.D. in computer science at the Illinois Institute of Technology. Abdur received his B.S. and M.S. in computer science at George Mason University in 1996 and 1994. He's authored many papers on process migration, fault-tolerant routing protocols and information retrieval topics. You can reach him at **abdur@cs.iit.edu**.

Shriman Gurung designs and implements global networking services for a software house based in London. In his spare time he runs a consulting firm offering help to start-ups. Shriman can be reached at **shriman@venus.co.uk**.

Don Kuenz works at Computing Resources Company (**http://gtcs.com/crc**). They provide programming, administration and hardware for Sun and PC platforms. You can reach Don at **kuenz@gtcs.com**.

Boris Loza holds a Ph.D. in computer science from Russia. He worked as a UNIX administrator and developer for 10 years. Currently he's working for Fidelity Investments Canada in the position of data security and capacity planner, doing IT security for UNIX, Windows NT and Novell. He has a daughter Anna and likes reading computer and mystery books and watching movies. He can be reached at **Boris.Loza@FMR.com**.

Rob Thomas is a systems, network, and security architect with the professional services division of a large telco. He can be reached at **robt@cymru.com**, or visit his home page at **www.enteract.com/~robt**.

Inside Solaris
Tips & Techniques for users of Sun Solaris

Inside Solaris (ISSN 1081-3314) is published monthly by Element K Journals, a division of Element K Press, 500 Canal View Boulevard, Rochester, N.Y., 14624.

Customer Relations

U.S. toll free	(800) 223-8720
Outside of the U.S.	(716) 240-7301
Customer Relations fax	(716) 214-2386

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

Element K Journals Customer Relations
500 Canal View Boulevard
Rochester, NY 14623

Or contact Customer Relations via Internet email at journals@element-k.com.

Editorial

Editor	Garrett Suhm
Assistant Editor	Jill Suhm
Managing Editor	Michelle Rogers
Assistant Managing Editor	Dianne Galloway
Copy Editors	Rachel Krayer
	Glenna Lechner
Contributing Editors	Abdur Chowdhury
	Shriman Gurung
	Don Kuenz
	Boris Loza
	Rob Thomas
Graphic Designer	Rachel J. King
Cover and Content Design	Melissa Ribaudo

You may address tips, special requests, and other correspondence to

The Editor, *Inside Solaris*
500 Canal View Boulevard
Rochester, NY 14623

Editorial Department fax	(716) 272-0064

Or contact us via Internet email at inside_solaris@elementkjournals.com.

Sorry, but due to the volume of mail we receive, we can't always promise a reply, although we do read every letter.

Element K Journals

General Manager	Kelly Baptiste
Manager of Customer Relations	Nicole Pate
Manager of Operations	Cristal Haygood
Manager of Graphic Design	Ian Caspersson
Manager of Product Marketing	Mike Mayfield
Senior Product Marketing Manager	Brian Cardona

Postmaster

Periodicals postage paid in Rochester, N.Y., and additional mailing offices.

Postmaster: Send address changes to

Inside Solaris
P.O. Box 92880
Rochester, NY 14692

Price

Domestic	$129/yr ($11.00 each)
Outside U.S.	$149/yr ($13.00 each)

Our Canadian GST# is: R140496720. CPM# is: 1446703. QST# is: 1018491237.

Back Issues

To order a back issue from the last six months, call Customer Relations at (800) 223-8720. Back issues cost $11.00 each, $13.00 outside the U.S. You can pay with MasterCard, VISA, Discover, or American Express.

Are you moving?

If you've moved recently or you're planning to move, you can guarantee uninterrupted service on your subscription by calling us at (800) 223-8720 and giving us your new address. Or you can fax us your label with the appropriate changes at (716) 214-2386. Our Customer Relations department is also available via email at journals@element-k.com.

Coming up

- Tuning Solaris for FireWall-1
- Security: threats and mistakes
- OCR for Solaris

the arguments entered when you invoke the script. At a minimum, this includes the host name of your upstream news server. It also may hold a username followed by a password, if your news server requires authentication. The next few lines in the script set variables to the directories used by our local news server. They reflect a default installation of INN.

The script starts invoking binary programs about one third of the way down. First it checks to see if a pull job is already running by looking for a lock file. When it finds a lock file it aborts; otherwise, it creates a lock file and continues. Next, it makes sure that it can talk to both the local news server and the upstream news server. If everything connects, it then pulls articles from your upstream news server. Next, it posts any articles originating from your local news server to the upstream server. Finally, it removes the lock file. After you finish tailoring the get.news.inn script to your own host, you can start pulling and posting news.

Pulling and posting news

You need to find out one more piece of information before you actually start pulling and posting news. Does your upstream news server authenticate users by asking for a username and password? Many news servers now authenticate users to make sure that no one abuses the server.

If your server requires authentication, use the following command:

```
get.news.inn news.isp.net \
  -U username -P password
```

On the other hand, if your news server doesn't require authentication use this command:

```
get.news.inn news.isp.net
```

Listing A shows the results displayed by our get .news.inn script during a successful news pull. You can see that we pulled in one article from comp.unix .solaris and another article from alt.solaris.x86. Look for additional messages when you post articles to your upstream news server.

Listing A: *Results displayed from a successful news pull*

```
Attempting to connect to news.upstream.net
Using Port 119
Official host name: news.upstream.net
Address: 192.168.1.1
Connected to news.upstream.net
200 News Service Ready (Typhoon v1.2.2) Posting Allowed
Loading active file from localhost
Reading current sucknewsrc
comp.unix.solaris - 1 articles 228115-228115
alt.solaris.x86 - 1 articles 112645-112645
Adding new groups from local active file to sucknewsrc
Elapsed Time = 0 mins 1.78 seconds
2 Articles to download
Deduping Elapsed Time = 0 mins 0.00 seconds
Deduped, 2 items remaining, 0 dupes removed.
Processing History File Elapsed Time = 0 mins 2.22 seconds
Processed history, 0 dupes removed
Total articles to download: 2
3918 Bytes received in 0 mins 1.05 secs, BPS = 3724.2
Closed connection to news.upstream.net
Posting Messages to localhost
2 Messages Posted
Elapsed Time = 0 mins 0.27 seconds
Cleaning up after myself
Downloaded Articles
```

Filtering your news

A few people abuse usenet by posting inappropriate articles, such as spam and off-topic messages. Running your own news server gives you access to industrial-strength news filters, which enable you to kill unwanted articles. Suck includes its own filtering functionality, which becomes particularly useful if your ISP bills by minutes connected or bytes downloaded.

Suck enables filtering when it finds a file with a default name of suckkillfile. In our example, suck looks for suckkillfile in the /user/local/news/db directory. Many times, abusers post a single news article to multiple, unrelated news groups. You can tell suck to kill all news articles cross posted to four or more newsgroups by placing a line similar to the following in the suckkillfile:

```
NRGRPS=4
```

✳