An Introduction to Oracle Solaris 11

Author: Pavel Anni, Oracle

pavel.anni@oracle.com

July 2012

Lab Introduction

In this lab we introduce the most interesting features of Solaris 11, based on real life use cases. We will:

- create a pool and a file system with ZFS; expand it; snapshot and clone it; use ZFS compression and deduplication;
- create a new boot environment as a backup, make our current system unbootable because of some fatal mistakes, reboot the system using backup BE;
- create a couple of zones; install some applications into them; clone a zone; use Resource Management with the zones;
- learn some Solaris security features which help to protect your systems and applications;
- use DTrace to find bottlenecks in the system

Prerequisites

This lab requires access to Solaris 11 system either in VirtualBox virtual machine or in Oracle Solution Center cloud.

Using the Oracle Solution Center cloud installation

Use your OSC instructions. You will be provided with IP addresses, access instructions and login credentials by your OSC instructor.

Using Your Machine

If you're performing this lab on your own machine, we've provided a VirtualBox appliance that contains all the software and configuration necessary to complete this lab. All you need is VirtualBox software and a modern (with a CPU supporting virtualization turned on, AMD-V or VT-x) laptop/desktop with at least 3 GB of RAM and 10GB of free disk space.

- Download and install the latest version of VirtualBox for your platform (http://www.virtualbox.org/wiki/Downloads).
- Download and install the latest version of VirtualBox Extensions Pack (http://www.virtualbox.org/wiki/Downloads).
- Download or find on the provided DVD and import the Hands On Lab machine into VirtualBox (File > Import Appliance). You will have to accept the OTN Oracle Solaris license to use the appliance.

The Environment

In this lab we are going to use Oracle Solaris 11 virtual appliance in Oracle VirtualBox environment. If you are using lab machines, the appliance is already installed. You can also download the appliance from Oracle Technology Network.

The Hands On Lab virtual appliance contains the following:

- Solaris 11 installation configured for this lab
- This lab document

By default, VirtualBox assigns the IP address 10.0.2.15 to the Solaris global zone. We will be using also IP addresses 10.0.2.16 and 10.0.2.17 for local zones. As we are using VirtualBox in NAT (network address translation) mode, this shouldn't interfere with your outside network environment.

If you are using OSC virtual machines, you will be provided with IP addresses to use in this lab.

You should login into Solaris desktop with the following credentials:

Username: `lab` Password: `oracle1`

After logging in, open a terminal window and assume the root role:

```
lab@solaris:~$ su root
```

Password for root is: **`oracle2011`**. Don't use the dash "-" in this command: we are going to keep the user `lab`'s environment settings

Note: we don't recommend to log in as `root`. In Solaris 11 it is prohibited by default; `root` is only a role, not a login name.

Lab Outline

The labs are independent from each other, you can take them in any order. We just recommend doing all exercises in each lab in order.

#	Topic (Use Case)	Exercise
ZFS Lab		
Z.1	You have some disks to use for your new file system. Create a new disk pool and a file system on top of it.	ZFS Pools
Z.2	You have to create home directories for your users; use file system quota to limit their space.	ZFS File systems
Z.3	You are becoming low on your disk space. Add a couple more disks to your pool and expand your file system. What other ZFS features can help you to save space in the future?	ZFS Compression
Z.4	Users tend to keep a lot of similar files in their archives. Is it possible to save space by using deduplication?	ZFS Deduplication
Z.5	A user has accidentally deleted her file. How to restore it without getting to the backup?	ZFS Snapshots
Boot Environments Lab		
B.1	You want to make updates to the system, but you want to be able to return back to the previous state.	Boot Environments
IPS Lab		
P.1	Solaris 11 has a new packaging system. How to use it?	IPS Basics
P.2	How can the new packaging system be used to minimize downtime when updating your Solaris installation?	IPS and Boot Environments
Networking Lab		
N.1	Solaris 11 introduces new networking commands. What's new and what's the difference?	Networking Basics
N.2	You want to create several virtual network interfaces to use with your zones.	Network Virtualization
Virtualization Lab		
V.1	Your development team wants a separate environment to develop their application.	Zones
V.2	You have to install some application packages in the zone and create users.	Inside a Zone
V.3	Your development team wants a copy of this environment.	Zone Cloning
V.4	Your departments want to know how much resources do they use to pay their fair share for the datacenter infrastructure.	Zone Monitoring
V.5	You want to control the zones' resource usage. You want to assign certain amount of processing power to each zone.	Resource Management
Security Lab		
S.1	User `root` is anonymous and too powerful. You want to track who performs superuser actions in your system	Solaris RBAC and Privileges

S.2	You want to give your developers the right to run DTrace, but not other superuser rights.	A Closer Look at Solaris Privileges
S.3	You want to limit rights for some processes to only very basic rights.	Process Privileges and Rights
S.4	You want to eliminate the need to run certain daemons with root privileges. Also you want to give the right to start some daemons to non-root users.	SMF, Process Privileges and Authorizations
DTrace Lab		
D.1	You have noticed that system utilization is very high. How to find the process which consumes most of the resources?	DTrace CPU
D.2	You have noticed that free storage space has decreased dramatically and keeps decreasing very fast. Who is "eating" our disk space?	DTrace Disk

Putting It All Together

The whole idea of this lab is to show you some Solaris 11 features that can be used to create a cloud infrastructure based on Solaris. You have just created storage pools and filesystems -- think cloud storage. It was fast, it was simple, it was flexible. You have created and cloned Solaris zones with applications within them -- think cloud machine instances. You have monitored and managed zones resources -- think cloud elasticity, metering and chargeback. We didn't discuss in this lab Solaris network virtualization, Solaris security, Solaris package management and many other features which make Solaris 11 truly cloud-oriented operating system. Try and learn more about Solaris 11 features!

Final Notes

The virtual appliance we used in this lab is configured to be able to perform zone installation without network access. Namely, we've configured an internal repository with just a small subset of packages necessary for zone installations. If you are going to continue using this appliance with open network access, you will need to change the repository address to Oracle's standard Solaris repository.

```
root@solaris:~# pkg set-publisher -G '*' -M '*' -g http://pkg.oracle.com/solaris/release -P solaris
```

Further Oracle Solaris Education

This Hands-on Lab is just and introduction in Oracle Solaris 11 world. We highly recommend to continue your education with Oracle University. There is a full set of new courses covering Oracle Solaris 11:

- Transition to Oracle Solaris 11
- What's New in Oracle Solaris 11
- What's New in Oracle Solaris 11 (Self-Study)
- Oracle Solaris 11 System Administration
- Oracle Solaris 11 Advanced System Administration

Get more details at the Oracle University page: http://bit.ly/OracleSolaris11Edu .

Good luck!

Oracle Solaris 11 ZFS Lab

Table of Contents

Exercise Z.1: ZFS Pools

Task: You have several disks to use for your new file system. Create a new disk pool and a file system on top of it.

Lab: We will check the status of disk pools, create our own pool and expand it.

Our Solaris 11 installation already has a ZFS pool. It's your root file system. Check this:

```
root@solaris:~# zpool list

NAME    SIZE  ALLOC   FREE  CAP  DEDUP  HEALTH  ALTROOT
rpool  15.9G  5.64G  10.2G  35%  1.00x  ONLINE  -
```

What do we know about this pool?

```
root@solaris:~# zpool status rpool
  pool: rpool
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        rpool       ONLINE       0     0     0
          c3t0d0s0  ONLINE       0     0     0

errors: No known data errors
```

Let's now create our own ZFS pool. What do we need for that? Just several disks and one command. We will create several files in /dev/dsk directory; they will act as disks in our lab:

```
root@solaris:~# cd /dev/dsk
root@solaris:~# mkfile 200m disk0 disk1 disk2 disk3 disk4 disk5 disk6 disk7 disk8 disk9
```

We'll take four disks and create a ZFS pool with RAID-Z protection:

```
root@solaris:~# zpool create labpool raidz disk0 disk1 disk2 disk3
```

That was easy, wasn't it? And fast, too! Check our ZFS pools again:

```
root@solaris:~# zpool list
NAME      SIZE  ALLOC   FREE  CAP  DEDUP  HEALTH  ALTROOT
labpool   780M   194K   780M   0%  1.00x  ONLINE  -
rpool    15.9G  7.57G  8.30G  47%  1.00x  ONLINE  -
```

By the way, the file system was also created and mounted automatically:

```
root@solaris:~# zfs list labpool
NAME                    USED  AVAIL  REFER  MOUNTPOINT
labpool                97.2K   551M  44.9K  /labpool
```

Do you need more space? Adding disks to the existing ZFS pool is as easy as creating it:

```
root@solaris:~# zpool add labpool raidz disk4 disk5 disk6 disk7
```

Check it again:

```
root@solaris:~# zfs list labpool
NAME      USED  AVAIL  REFER  MOUNTPOINT
labpool  97.2K  1.11G  44.9K  /labpool
```

Note the increased file system's size.

Back to top

Exercise Z.2: ZFS File Systems

Task: You have to create home directories for your users; use file system quota to limit their space.

Lab: We'll create a user "`joe`" and set a disk quota for him.

Creating a user is pretty similar to most Unix/Linux systems. What's different is what's going on behind the scenes.

```
root@solaris:~# useradd -m joe
root@solaris:~# passwd joe
New Password: oracle1
Re-enter new Password: oracle1
passwd: password successfully changed for joe
```

In Solaris 11 behind the scenes we create a *separate* ZFS file system for the user (parameter `-m`) in `/export/home` and mount it with AutoFS service at `/home/`*`username`* when the user logs in. Check it:

```
root@solaris:~# zfs list
NAME                      USED  AVAIL  REFER  MOUNTPOINT
labpool                  97.2K  1.11G  44.9K  /labpool
rpool                    7.65G  7.97G    39K  /rpool
rpool/ROOT               5.59G  7.97G    31K  legacy
rpool/ROOT/solaris       5.59G  7.97G  5.17G  /
rpool/ROOT/solaris/var    330M  7.97G   183M  /var
rpool/dump               1.03G  8.01G  1.00G  -
rpool/export             1.48M  7.97G    32K  /export
rpool/export/home        1.44M  7.97G    33K  /export/home
rpool/export/home/joe     686K  7.97G   686K  /export/home/joe
rpool/export/home/lab     760K  7.97G   760K  /export/home/lab
rpool/swap               1.03G  8.01G  1.00G  -
```

What does it mean for us, system administrators? That means we can use all kinds of ZFS features (compression, deduplication, encryption) on a per-user basis. We can create snapshots and perform rollbacks on a per-user basis. More about that later. Now we'll set a disk quota for `joe`'s home directory.

```
root@solaris:~# zfs set quota=200m rpool/export/home/joe
```

Now change user to "`joe`" and check how much space you can use:

```
root@solaris:# su - joe
joe@solaris$ mkfile 110m file1
joe@solaris$ mkfile 110m file2
```

First file was created OK, but with the second one we've got an error: "Disk quota exceeded".

Change the quota for `joe` in the other window:

```
root@solaris:~# zfs set quota=300m rpool/export/home/joe
```

Then change back to the `joe`'s window and try again:

```
joe@solaris$ /usr/sbin/mkfile 110m file2
```

Success! As you can see, it's pretty easy to create and manage ZFS filesystems. Remember, by default Solaris 11 creates a separate ZFS file system for each user.

Back to top

Exercise Z.3: ZFS Compression

Task: You are becoming low on your disk space. Now you know how to add more disks to your pool and expand your file system. What other ZFS features can help you to solve this problem?

Lab: In our lab we will compress our Solaris manuals directory and see if we are able to use it after that. Create a separate filesystem for this:

```
root@solaris:~# zfs create rpool/zman
root@solaris:~# zfs list | grep zman
rpool/zman                 31K  7.78G    31K  /rpool/zman
```

Set compression to "gzip" (there are options to gzip and other algorithms too--check the manual).

```
root@solaris:~# zfs set compression=gzip rpool/zman
```

Copy our Solaris manuals there (it will take some time, be patient):

```
root@solaris:~# cp -rp /usr/share/man/* /rpool/zman/
```

Compare the sizes:

```
root@solaris:~# du -sh /usr/share/man /rpool/zman
 149M   /usr/share/man
  68M   /rpool/zman
```

We just have saved about 55% of disk space. Not bad! Check if you are able to use the manuals after compression:

```
root@solaris:~# export MANPATH=/rpool/zman
root@solaris:~# man ls
```

Back to top

Exercise Z.4: ZFS Deduplication

Task: Users tend to keep a lot of similar files in their archives. Is it possible to save space by deduplication?

Lab: We will create a ZFS file system with deduplication turned on and see if it helps.

Let's model the following situation: we have a file system which is used as an archive. We'll create separate file systems for each user and imagine that they store similar files there.

Remember we have created ZFS pool called `labpool` in the first exercise? If you have skipped that exercise, create it now:

```
root@solaris:~# zpool create labpool raidz disk0 disk1 disk2 disk3
```

Create a file system with deduplication and compression:

```
root@solaris:~# zfs create -o dedup=on -o compression=gzip labpool/archive
```

Create users' file systems (we'll call them a, b, c, d for simplicity):

```
root@solaris:~# zfs create labpool/archive/a
root@solaris:~# zfs create labpool/archive/b
root@solaris:~# zfs create labpool/archive/c
```

```
root@solaris:~# zfs create labpool/archive/d
```

Check their "dedup" parameter:

```
root@solaris:~# zfs get dedup labpool/archive/a
NAME               PROPERTY  VALUE          SOURCE
labpool/archive/a  dedup     on             inherited from labpool/archive
```

Children file systems inherit parameters from their parents.

Create an archive from /usr/share/man, for example.

```
root@solaris:~# tar czf /tmp/man.tar.gz /usr/share/man
```

And copy it to four file systems we've just created. Don't forget to check deduplication rate after each copy.

```
root@solaris:~# cd /labpool/archive
root@solaris:/labpool/archive# ls -lh /tmp/man.tar.gz
-rw-r--r--   1 root     root         34M Nov 15 09:12 /tmp/man.tar.gz
root@solaris:/labpool/archive# zpool list labpool
NAME      SIZE  ALLOC   FREE  CAP  DEDUP  HEALTH  ALTROOT
labpool  1.52G  1.05M  1.52G   0%  1.00x  ONLINE  -
root@solaris:/labpool/archive# cp /tmp/man.tar.gz a/
root@solaris:/labpool/archive# zpool list labpool
NAME      SIZE  ALLOC   FREE  CAP  DEDUP  HEALTH  ALTROOT
labpool  1.52G  40.5M  1.48G   2%  1.00x  ONLINE  -
root@solaris:/labpool/archive# cp /tmp/man.tar.gz b/
root@solaris:/labpool/archive# zpool list labpool
NAME      SIZE  ALLOC   FREE  CAP  DEDUP  HEALTH  ALTROOT
labpool  1.52G  40.7M  1.48G   2%  2.00x  ONLINE  -
root@solaris:/labpool/archive# cp /tmp/man.tar.gz c/
root@solaris:/labpool/archive# zpool list labpool
NAME      SIZE  ALLOC   FREE  CAP  DEDUP  HEALTH  ALTROOT
labpool  1.52G  41.5M  1.48G   2%  3.00x  ONLINE  -
root@solaris:/labpool/archive# cp /tmp/man.tar.gz d/
root@solaris:/labpool/archive# zpool list labpool
NAME      SIZE  ALLOC   FREE  CAP  DEDUP  HEALTH  ALTROOT
labpool  1.52G  41.2M  1.48G   2%  4.00x  ONLINE  -
```

It might take a couple of seconds for ZFS to commit those changes and report the correct dedup ratio. Just repeat the command if you don't see the results listed above.

Remember, we set compression to "on" as well when we created the file system? Check the compression ratio:

```
root@solaris:/labpool/archive# zfs get compressratio labpool/archive
NAME             PROPERTY       VALUE  SOURCE
labpool/archive  compressratio  1.01x  -
```

The reason is simple: we placed in the file system files that are compressed already. Sometimes compression can save you some space, sometimes deduplication can help.

Back to top

Exercise Z.5: ZFS Snapshots

Task: A user has accidentally deleted her file. How to restore it without getting to the backup?

Lab: The following command shows all the file systems with their snapshots:

```
root@solaris:~# zfs list -r -t all rpool
NAME                              USED  AVAIL  REFER  MOUNTPOINT
```

```
rpool                          7.72G  7.92G    40K  /rpool
rpool/ROOT                     5.59G  7.92G    31K  legacy
rpool/ROOT/solaris             5.59G  7.92G  5.17G  /
rpool/ROOT/solaris@install     96.9M      -  2.99G  -
rpool/ROOT/solaris/var          330M  7.92G   183M  /var
rpool/ROOT/solaris/var@install  147M      -   302M  -
rpool/dump                     1.03G  7.95G  1.00G  -
rpool/export                    864K  7.92G    32K  /export
rpool/export/home               832K  7.92G    34K  /export/home
rpool/export/home/joe            35K  7.92G    35K  /export/home/joe
rpool/export/home/lab           763K  7.92G   763K  /export/home/lab
rpool/swap                     1.03G  7.95G  1.00G  -
rpool/zman                     63.0M  7.92G  63.0M  /rpool/zman
```

As you can see, in the freshly installed system we already have a snapshot of our original installation (`solaris@install`). Let's see what can be done with snapshots. Return to the `lab`'s home directory and create a sample text file `first.txt` with a text editor (`gedit, vi`) or with a simple Solaris command:

```
root@solaris:~# cd /home/lab
root@solaris:~# echo "first line\nsecond line" > first.txt
root@solaris:~# cat first.txt
first line
second line

root@solaris:~# zfs snapshot rpool/export/home/lab@snap1
root@solaris:~# zfs list -r -t all rpool
NAME                            USED  AVAIL  REFER  MOUNTPOINT
rpool                          7.72G  7.92G    40K  /rpool
rpool/ROOT                     5.59G  7.92G    31K  legacy
rpool/ROOT/solaris             5.59G  7.92G  5.17G  /
rpool/ROOT/solaris@install     96.9M      -  2.99G  -
rpool/ROOT/solaris/var          330M  7.92G   183M  /var
rpool/ROOT/solaris/var@install  147M      -   302M  -
rpool/dump                     1.03G  7.95G  1.00G  -
rpool/export                    864K  7.92G    32K  /export
rpool/export/home               832K  7.92G    34K  /export/home
rpool/export/home/joe            35K  7.92G    35K  /export/home/joe
rpool/export/home/lab           763K  7.92G   763K  /export/home/lab
rpool/export/home/lab@snap1        0      -   763K  -
rpool/swap                     1.03G  7.95G  1.00G  -
rpool/zman                     63.0M  7.92G  63.0M  /rpool/zman
```

The snapshot uses 0 bytes because we have not changed anything in your home directory. Add a couple of sentences to our file "`first.txt`" and save it again.

```
root@solaris:~# echo "third line\nfourth line" >> first.txt
```

How does it look like now?

```
root@solaris:~# cat first.txt
first line
second line
third line
fourth line
```

What if we wanted to restore our old content? Just roll it back:

```
root@solaris:~# zfs rollback rpool/export/home/lab@snap1
```

Check the file:

```
root@solaris:~# cat first.txt
first line
second line
```

It's simple, it's fast, it doesn't require much space. Actually, it doesn't require space at all until you have made some changes.

It should be noted that if you want to rollback not the latest snapshot, all more recent snapshots will be deleted (Solaris will warn you about that). Obviously, it's possible to make snapshots on a regular basis. You don't even have to use `cron(1M)` for that, there is a special package called `zfs-auto-snapshot`.

Food for thought: How can snapshots be used in the real life environment? Backup is the first idea that comes to mind. What else?

Back to top

Oracle Solaris 11 Boot Environments Lab

Exercise B.1: Boot Environments

Task: You want to make updates to your system, but you want to be able to return back to the previous state.

Lab: We will use a Solaris 11 feature called Boot Environments. We'll create an extra boot environment as a backup (think saving your state in a shooting game). Then we'll make some fatal mistakes which make our system unbootable. After we failed to boot our default boot environment, we'll boot into the backup BE.

The only command you want to know to work with Boot Environments is `beadm(1M)`. Start with showing all boot environments in the system:

```
root@solaris:~# beadm list
BE      Active Mountpoint Space Policy Created
--      ------ ---------- ----- ------ -------
solaris NR     /          6.87G static 2011-11-14 13:13
```

Create a new boot environment:

```
root@solaris:~# beadm create solaris-backup
```

Check the status again:

```
root@solaris:~# beadm list
BE           Active Mountpoint Space   Policy Created
--           ------ ---------- -----   ------ -------
solaris        NR    /          7.13G  static 2011-11-14 13:13
solaris-backup      -        126.14M  static 2011-11-15 11:09
```

Now pretend you are making some changes in the system configuration, creating and removing directories, and... Somebody has distracted you and instead of removing a temporary directory, you have typed:

```
root@solaris:~# rm -rf /etc/
```

What??? You just have destroyed the whole /etc directory! All the configuration files are gone! (If you are brave enough, you can remove /usr/bin too!) You have killed your system! Try to "Power Off" your virtual machine and reboot it--you'll see it won't boot. Don't wait too long, 3-5 minutes is enough to be sure that it doesn't boot.

No worries! We have a backup! And you don't have to go and find the backup tape in your fireproof cabinet, go through all the hassles of restoring the unbootable system... Just when your VirtualBox VM shows you the GRUB menu, choose the "solaris-backup" item instead of the "Oracle Solaris 11 11/11" which is the default.

It boots again! What a relief!

Boot Environments have many applications: you can update your system, installing packages into inactive boot environment; create boot environment snapshots etc. We leave it for your homework.

Back to top

Oracle Solaris 11 IPS Lab

One of the main new features in Solaris 11 is a new packaging system, called IPS (Image Packaging System). In this lab we will explore its capabilities and learn how to work with packages from System Administrator's perspecitve.

Exercise P.1: IPS Basics

Task: You want to find and install a package from Solaris repository.

Lab: We will learn some basic IPS commands used to look for a package, inquire about its content etc. When we found the package we needed, we install it.

Let's imagine we want to do some network load testing, so we want to run the utility called 'iperf'. Try to run this command and find out that it's not installed:

```
root@solaris:~# iperf
```

So the first thing we do is show our current publisher. The publisher is where the IPS repository is located. It can be a local directory, an NFS mount point, an internal http ot https server, or an Internet repository like http://pkg.oracle.com/solaris. You system can have several publishers configured.

```
root@solaris:~# pkg publisher
```

OK, it seems we are going to use our local publisher installed in our Solaris system.

Now we list our installed packages

```
root@solaris:~# pkg list |more
```

Let's see how many packages are installed

```
root@solaris:~# pkg list |wc
```

Now let's see how many packages are available in the repository

```
root@solaris:~# pkg list -a |wc
```

As you can see, our local repository is pretty small, it contains only the packages we need for this lab. If you have changed the publisher to http://pkg.oracle.com/solaris/release as it's described in the introduction, you would see many more packages.

Let's now do a local search for iperf (among the installed packages). It will find nothing:

```
root@solaris:~# pkg search -l iperf
```

Now do the same search, without the -l flag so it goes to the repository:

```
root@solaris:~# pkg search iperf
```

Next we get some information about the package like date of creation, version number, etc.

```
root@solaris:~# pkg info -r iperf
```

And we can see exactly what files make up the package:

```
root@solaris:~# pkg contents -r iperf
```

```
root@solaris:~# pkg search benchmark/iperf:depend::
```

And now we install the `iperf` package:

```
root@solaris:~# pkg install iperf
```

Here we demonstrate what kind of metadata is kept in IPS:

```
root@solaris:~# pkg contents -t file -o owner,group,mode,pkg.size,path iperf
```

Now we run iperf to show it's now found:

```
root@solaris:~# iperf
```

Show how we can verify a package has not been compromised:

```
root@solaris:~# pkg verify iperf
```

Let's check the permission bits of `iperf`:

```
root@solaris:~# ls -l /bin/iperf
```

Let's say someone changed the permissions bits like this:

```
root@solaris:~# chmod 777 /bin/iperf
```

Let's verify again, and it will return an error:

```
root@solaris:~# pkg verify iperf
```

Now let's fix this package:

```
root@solaris:~# pkg fix iperf
```

The next few lines show an interesting way of getting a package name from a arbitrary file. Let's take `vi` editor for test. We first get a SHA1 digest of '`/usr/bin/vi`'

```
root@solaris:~# digest -a sha1 /usr/bin/vi
```

Since the digest is saved in the package DB, we can search for that hash and see what matches it:

```
root@solaris:~# pkg search -l f2495fa19fcc4b8a403e0bd4fef809d031296c68
```

How can we use it? Imagine someone has renamed some important file to hide his tracks. Using this methid we can find out the original name of the file.

Now the command we use to update our installed packages:

```
root@solaris:~# pkg update
```

It doesn't show anything as we are using the original "release" repository. If we had the "support" repository configured, it would find all the updates and install them.

Lastly, we show how you can view a history of package commands:

```
root@solaris:~# pkg history
```

And a verbose history:

```
root@solaris:~# pkg history -l|more
```

And finally, we uninstall the `iperf` package:

```
root@solaris:~# pkg uninstall iperf
```

Exercise P.2: IPS and Boot Environments

Another important feature of IPS is the ability to perform all package related operations not only on the current boot environment, but also on a mounted one. Imagine you want to update your system, but you want to keep your current state untouched to be able to return back to safety in case something goes wrong. Also you want to minimize the downtime.

Create a new boot environment for the updated system:

```
root@solaris:~# beadm create solaris-updated
```

In real life you might want to use some naming policy for the BEs, like timestamping them.

Now mount this boot environment in your file system:

```
root@solaris:~# beadm mount solaris-updated /mnt
```

Now you can perform any package operations with this mounted boot environment. As we don't have updates in our repository, we just install a package (the same iperf package), check that it's not available in our current BE and then reboot the system with the updated BE and make sure the package is installed there.

```
root@solaris:~# pkg -R /mnt install iperf
```

Now make sure the new boot environment is active on reboot:

```
root@solaris:~# beadm activate solaris-updated
```

Then reboot the system and check if iperf is available.

Imagine how much downtime you can save when using this method to update your system!

Oracle Solaris 11 Networking Lab

In Solaris 11 several new networking commands were added, some management practices have changed to make network administration easier and more robust. In this lab we will learn some basic networking commands, compare them to the old ones and also work with network virtualization features, which are brand new in Solaris 11.

Exercise N.1: Solaris 11 Networking Basics

Task: You have to configure network interfaces and network services (DNS) in Solaris.

Lab: We will configure network interfaces in manual mode. We will use the new way of configuring DNS and also we will import the old configuration. We will use a new feature called Vanity Naming with allows you to give network interfaces any names you want. Note that when we use these new Solaris 11 commands, all the changes are persistent and will sustain a reboot.

First, we'll change network management to Manual mode:

```
root@solaris:~# netadm enable -p ncp DefaultFixed
```

Show available physical network interfaces:

```
root@solaris:~# dladm show-phys
```

Create an interface with static IPv4 address:

```
root@solaris:~# ipadm delete-ip net0
root@solaris:~# ipadm create-ip net0
root@solaris:~# ipadm show-if
root@solaris:~# ipadm create-addr -T static -a local=10.9.8.7/24 net0/addr
root@solaris:~# ipadm show-addr
```

Create an interface and get the address from DHCP:

```
root@solaris:~# ipadm delete-ip net0
root@solaris:~# ipadm create-ip net0
root@solaris:~# ipadm create-addr -T dhcp net0/addr
root@solaris:~# ipadm show-addr
```

Create an interface with auto-generated IPv6 configuration:

```
root@solaris:~# ipadm delete-ip net0
root@solaris:~# ipadm create-ip net0
root@solaris:~# ipadm create-addr -T addrconf net0/addr
root@solaris:~# ipadm show-addr
```

Configure the default route:

```
root@solaris:~# route -p add default 10.0.2.2
```

Activate DNS configuration:

```
root@solaris:~# svccfg -s dns/client \'setprop config/nameserver = net_address: 192.168.1.1\'
root@solaris:~# svccfg -s dns/client \'setprop config/domain = astring: \"example.com\"\'
root@solaris:~# svccfg -s name-service/switch \'setprop config/host = astring: \"files dns\"\'
root@solaris:~# svcadm refresh name-service/switch
root@solaris:~# svcadm refresh dns/client
```

Alternatively, you can edit the usual files resolv.conf and nsswitch.conf, but you have to import them into the naming service configuration:

```
root@solaris:~# nscfg import -f svc:/system/name-service/switch:default
root@solaris:~# nscfg import -f svc:/network/dns/client:default
```

```
root@solaris:~# svcadm refresh dns/client
```

Do you remember the days when you were a junior Solaris system administrator and wondered why all network interfaces in Solaris have these funny names? `le, bge, ce, xge, e1000g...`. Now, as you can see, they all are called net0, net1, net2, ... Much simpler, right? Even more than that: you can give your interfaces your own names. Here is the example. Show what we've got now:

```
root@solaris:~# dladm show-phys
root@solaris:~# ipadm show-addr
```

We now decide to rename net0 to foo1. Start by deleting the net0 IP interface

```
root@solaris:~# ipadm delete-ip net0
```

...now rename the NIC

```
root@solaris:~# dladm rename-link net0 foo1
root@solaris:~# dladm show-phys
```

Add back in the IP interface

```
root@solaris:~# ipadm create-ip foo1
```

And put it under DHCP control like it was before

```
root@solaris:~# ipadm create-addr -T dhcp foo1/dhaddr
```

```
root@solaris:~# ipadm show-addr
```

Cleaning up... Undo it all

```
root@solaris:~# ipadm delete-ip foo1
root@solaris:~# dladm rename-link foo1 net0
root@solaris:~# ipadm create-ip net0
root@solaris:~# ipadm create-addr -T dhcp net0/dhaddr
root@solaris:~# ipadm show-addr
```

One word of advice: having this kind of freedom, please try to avoid long discussions about network interface naming, similar to what you have already had regarding host naming policies. :-)

Exercise N.2: Network Virtualization

Task: You want to create Virtual Network Interface Cards (VNICs) to use them with your Zones. You want to build and manage your application's network infrastructure completely inside the box for development and testing purposes.

Lab: We will create VNICs, assign IP addresses to them and learn how to limit bandwidth on them.

First we show the links. Links can be physical or virtual. Note that for physical NICs, we use a new naming scheme net0, net1, etc. that hides the actual device name.

```
root@solaris:~# dladm show-link
```

Show only the physical ethernet NICs:

```
root@solaris:~# dladm show-ether
```

And to see the actual hardware devices used for the netX NICs:

```
root@solaris:~# dladm show-phys
```

The next command shows a bit more information like the physical location:

```
root@solaris:~# dladm show-phys -L
```

So now we create a VNIC that we call vnic1, using net0 as it's interface Note that VNICs are first-class NICs in terms of visibility (e.g. snoop)

```
root@solaris:~# dladm create-vnic -l net0 vnic1
```

Show the VNICs:

```
root@solaris:~# dladm show-vnic
```

Show how easy it is to limit bandwith on a VNIC:

```
root@solaris:~# dladm set-linkprop -p maxbw=40 vnic1
root@solaris:~# dladm show-vnic
```

Now we create an IP interface. This is analgous to plumbing the interface:

```
root@solaris:~# ipadm create-ip vnic1
```

Now we assign a persistent IP address to the VNIC:

```
root@solaris:~# ipadm create-addr -T static -a 10.2.3.4 vnic1/v4static
```

Ping the VNIC:

```
root@solaris:~# ping 10.2.3.4
```

Show all available datalinks, both physical and virtual

```
root@solaris:~# dladm show-link
```

If you want to know more about the internals of these commands, here is how to show the "database" files where the dladm and ipadm commands keep their persistent data:

```
root@solaris:~# ls -lrt /etc/dladm /etc/ipadm
```

But remember: you are not supposed to edit these files manually, always use `dladm` and `ipadm` commands!

Finally list all IP addresses:

```
root@solaris:~# ipadm show-addr
```

Now we tear down what we've just created:

```
root@solaris:~# ipadm delete-addr vnic1/v4static
root@solaris:~# ipadm delete-ip vnic1
root@solaris:~# dladm delete-vnic vnic1
root@solaris:~# dladm show-link
```

Now you see how new networking commands work. Of course, you can still use the old-style `ifconfig`, but the new commands are easier to use and, most importantly, they make presistent changes.

Find more food for thought and inspiration here:

- How to Script Oracle Solaris 11 Zones Creation for a Network-In-a-Box Configuration
  http://www.oracle.com/technetwork/articles/servers-storage-admin/o11-118-s11-script-zones-524499.html
- How to Restrict Your Application Traffic Using Oracle Solaris 11 Network Virtualization and Resource Management
  http://www.oracle.com/technetwork/articles/servers-storage-admin/o11-095-s11-app-traffic-525038.html

Oracle Solaris 11 Virtualization Lab

Table of Contents

Exercise V.1: Zones

Task: Your development team wants a separate environment to develop their new application.

Lab: We are going to use Solaris virtualization technology called Solaris Zones. First, we have to create a filesystem where all the zones will be located. In Solaris 11 it must be a ZFS filesystem.

```
root@solaris:~# zfs create -o compress=gzip -o dedup=on -o mountpoint=/zones rpool/zones
```

(Now we know how to save space with ZFS compression and dedup options!)

Check if it's created and mounted:

```
root@solaris:~# zfs list rpool/zones
NAME         USED  AVAIL  REFER  MOUNTPOINT
rpool/zones   31K  7.92G    31K  /zones
```

Create a zone with a minimum set of parameters: `name` (in our case it will be `zone1`) and `zonepath`, where the zone's files will be located.

```
root@solaris:~# zonecfg -z zone1
zone1: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:zone1> create
create: Using system default template 'SYSdefault'
zonecfg:zone1> set zonepath=/zones/zone1
zonecfg:zone1> exit
```

As simple as that! By default, the zone will be configured with Exclusive-IP (that means it will use its own IP stack) and a VNIC will be automatically created for the zone.

To check the status of our newly created zone:

```
root@solaris:~# zoneadm list -cv
  ID NAME             STATUS     PATH                           BRAND    IP
   0 global           running    /                              solaris  shared
   - zone1            configured /zones/zone1                   solaris  excl
```

The zone is configured, we can install and boot it right now. But before the installation we'll configure a profile for the Solaris instance which will be running inside the zone. We are trying to avoid configuring the zone interactively and have it ready for use right after the first boot.

```
root@solaris:~# sysconfig create-profile -o zone1-profile.xml
```

This command will bring you to the interactive dialog very similar to the standard Solaris installaion. You will have to enter:

- Computer Name (hostname for the zone): `zone1`
- Network configuration: choose "`Automatically`"
- Time zone: choose your time zone from the list
- Date: confirm the current date

- Root password: `oracle2011`
- New user account details: real name, login name and password. This will be the first user of the zone. We have entered "Zone User", "`zuser1`", "`oracle1`"

Now, when the zone's profile is created, we can install the zone and initialize it using this profile.

```
root@solaris:~# zoneadm -z zone1 install -c /home/lab/zone1-profile.xml
A ZFS file system has been created for this zone.
Progress being logged to /var/log/zones/zoneadm.20111113T200358Z.zone1.install
       Image: Preparing at /zones/zone1/root.

 Install Log: /system/volatile/install.4418/install_log
 AI Manifest: /tmp/manifest.xml.NVaaNi
  SC Profile: /tmp/zone1-profile.xml
    Zonename: zone1
Installation: Starting ...
```

Here you can take a break. The installation will take about 10 minutes, depending on your network connection.

```
 ...Long output is skipped...
Next Steps: Boot the zone, then log into the zone console (zlogin -C)
to complete the configuration process.
```

Check the status again:

```
root@solaris:~# zoneadm list -cv
  ID NAME             STATUS     PATH                           BRAND    IP
   0 global           running    /                              solaris  shared
   - zone1            installed  /zones/zone1                   solaris  excl
```

It's time to boot our zone:

```
root@solaris:~# zoneadm -z zone1 boot
root@solaris:~# zoneadm list -cv
  ID NAME             STATUS     PATH                           BRAND    IP
   0 global           running    /                              solaris  shared
   1 zone1            running    /zones/zone1                   solaris  excl
```

Note the zone's status has changed to "running".

Now log into our zone's console (note -C). You will have to wait a couple of minutes while the system is initializing services for the first time.

```
root@solaris:~# zlogin -C zone1
[Connected to zone 'zone1' console]
```

You will get the standard Solaris login prompt. Congratulations! You've just configured "virtualization within virtualization" using Oracle technologies: Solaris zones within Oracle VirtualBox.

Try to login using `root`'s credentials (`root/oracle2011`). Here is the result:

```
zone1 console login: root
Password:
Roles can not login directly
Login incorrect
Nov 13 15:23:07 zone1 login: login account failure: Permission denied
```

A-ha! This is a new Solaris 11 security feature called "root as a role". That means that you can't login into a system as "root". You have to use normal user's credentials and only then you will be able to use "sudo" or "pfexec" according to your roles and privileges.

Try to login again with `zuser1/oracle1`.

```
Oracle Corporation      SunOS 5.11      11.0    November 2011
zuser1@zone1:~$
```

Success!

Note: to escape from the zone's console use: ~. (tilde period).

Back to top

Exercise V.2: Inside the Zone

Task: You have to install some application packages in the zone and create some users.

Lab: Log in in the zone, create a user and install a web server application.

```
root@solaris:~# zlogin zone1
root@zone1:~#
```

Play around with the usual sysadmin commands. How can you tell if you are in a zone or not? First, try `ps -ef`. Do you see anything unusual? Yes, you are right, the process IDs don't start with 0, but with some big number. Other than that, no visible difference between the normal Solaris installation and the zone. Try `uname -a, psrinfo, cat /etc/release` ...

Now let's do something useful with the zone. Like running a web server, for example. Let's install and run Apache.

```
root@zone1:~# pkg list -a *apache*
. . .Skipped. . .
web/server/apache-22 2.2.16-0.151.0.1 known ----.
. .Skipped. . .
root@zone1:~# pkg install apache-22
. . .Skipped. . .
```

We've installed it successfully, but it's not running yet.

```
root@zone1:~# svcs -a | grep apache
disabled 6:31:42 svc:/network/http:apache22
```

Start the Apache web server:

```
root@zone1:~# svcadm enable apache22
root@zone1:~# svcs -a | grep apache
online 6:34:03 svc:/network/http:apache22
```

Check if it's working from your global Solaris zone (your Solaris desktop): start Firefox and enter your zone's IP address into the URL field: 10.0.2.16. "It works!" -- the page usually reads. In our current VirtualBox configuration (with NAT networking) the zone is not visible from outside, but you can always try to change your VirtualBox configuration to Bridged networking and give your zone an IP address from your local network. (Do try this at home!).

Check if it's your zone who is talking. Go back to the zone's terminal window and change your web server homepage (I'm using vi here, as we don't have many choices in a freshly installed zone. If you are not familiar with vi, check our Vi Quick Reference below):

```
root@zone1:~# vi /var/apache2/2.2/htdocs/index.html
```

Write here something like "This is Zone1 and it works!", save the file and reload the page in Firefox in your Solaris desktop. Did it work? Congratulations!

```
Vi Quick Reference

If you're unfamiliar with vi, following are a few common
keyboard commands to get you through this exercise:
i = switch to Insert mode
```

```
Use Insert mode to type in your text.

Esc = switch to Command mode

In Command mode use:
k = up
j = down
w = right or forward one word
b = left or back one word
l = right 1 char
h = left 1 char
x = delete 1 char
u = undo
dd = delete entire current line
:w = write (save) the current file
:wq = write and quit
:w! = write to a read-only file
:q! = quite ignoring changes (do not write)
```

What else do we need? Try to create users in the zone.

```
root@zone1:~# useradd -m jack
root@zone1:~# passwd jack
New Password: oracle1 (will not be displayed)
Re-enter new Password: oracle1 (will not be displayed)
passwd: password successfully changed for jack
root@zone1:~# su - jack
Oracle Corporation      SunOS 5.11      11.0    November 2011
jack@zone1:~$ ls
local.cshrc    local.login    local.profile
jack@zone1:~$
```

Looks good! Try to login from your global zone (open another window on your Solaris desktop):

```
lab@solaris:~$ ssh -l jack 10.0.2.16
```

(It's a small letter L here, not the digit one)

For your homework: compare global and non-global zones installations. How many packages are installed in both? How many services are running? Check if you can login into the global zone with the zone user's (jack) credentials. Check if you can use your zone's root password in the global zone (of course, if they are different).

Back to top

Exercise V.3: Zone Cloning

Task: Your development team wants a copy of this environment for testing purposes.

Lab: We will configure a new zone ('zone2') and then clone it from the existing zone1.

This time let's configure the zone in one line:

```
root@solaris:~# zonecfg -z zone2 "create; set zonepath=/zones/zone2; exit"
```

Check:

```
root@solaris:~# zoneadm list -cv
  ID NAME             STATUS     PATH                           BRAND    IP
   0 global           running    /                              solaris  shared
   1 zone1            running    /zones/zone1                   solaris  excl
```

```
   - zone2            configured /zones/zone2                    solaris  excl
```

Before cloning we have to shutdown our running zone1:

```
root@solaris:~# zoneadm -z zone1 shutdown
```

Then we create the new zone's profile. Start the System Configuration Tool and repeat all the configuration steps you did for zone1. Just change Computer Name to "zone2", user name to "zuser2" and password to "oracle2".

```
root@solaris:~# sysconfig create-profile -o zone2-profile.xml
```

Now clone zone1 and configure zone2 automatically using this profile:

```
root@solaris:~# zoneadm -z zone2 clone -c /home/lab/zone2-profile.xml zone1
root@solaris:~# zoneadm list -cv
  ID NAME             STATUS     PATH                           BRAND    IP
   0 global           running    /                              solaris  shared
   1 zone1            installed  /zones/zone1                   solaris  excl
   2 zone2            installed  /zones/zone2                   solaris  excl
```

Now boot both zones:

```
root@solaris:~# zoneadm -z zone1 boot
root@solaris:~# zoneadm -z zone2 boot
root@solaris:~# zoneadm list -cv
  ID NAME             STATUS     PATH                           BRAND    IP
   0 global           running    /                              solaris  shared
   1 zone1            running    /zones/zone1                   solaris  excl
   2 zone2            running    /zones/zone2                   solaris  excl
```

Success! And it was faster than the initial installation, wasn't it?

After it's done, login into zone2.

root@solaris:~# `zlogin zone2`

First of all, what about our Apache server?

```
root@zone2:~# pkg list -a | grep apache
. . .Skipped. . .
web/server/apache-22                              2.2.20-0.175.0.0.0.2.537   i--
```

Great! It's installed already! Check if it's running:

```
root@zone2:~# svcs *apache*
online 11:48:47 svc:/network/http:apache22
```

Try the zone2 address (10.0.2.17) in Firefox in the global zone.

"This is Zone1 and it works!" - of course, we have cloned not only installed applications, but also their configurations. Change it to "Zone2", just for consistency sake.

```
root@zone2:~# vi /var/apache2/2.2/htdocs/index.html
```

Back to top

Exercise V.4: Zone Monitoring

Task: Your departments want to know how much resources do they use to pay their fair share for the datacenter infrastructure.

Lab: Some familiar Solaris commands now include a -Z parameter to help you to monitor zones behavior. Try ps -efZ and prstat -Z to take a look. Try also a new command zonestat to show zone statistics.

```
root@solaris:~# zonestat -z zone1,zone2 5

Collecting data for first interval...
Interval: 1, Duration: 0:00:05
SUMMARY                   Cpus/Online: 1/1   PhysMem: 2047M  VirtMem: 3071M
                     ---CPU----  --PhysMem-- --VirtMem-- --PhysNet--
                ZONE  USED %PART  USED %USED  USED %USED PBYTE %PUSE
            [total]  0.05 5.45%  968M 47.3% 1251M 40.7%     0 0.00%
           [system]  0.01 1.51%  287M 14.0%  735M 23.9%     -     -
              zone1  0.00 0.16% 73.8M 3.60% 66.3M 2.16%     0 0.00%
              zone2  0.00 0.13% 73.9M 3.61% 67.2M 2.18%     0 0.00%
```

Note the parameters you can observe with `zonestat`: CPU utilization, physical and virtual memory usage, network bandwidth utilization.

Back to top

Exercise V.5: Resource Management

Task: You want to control the zones' resource usage. You want to assign certain amount of processing power to each zone.

Lab: We now know how to create and clone zones. Now let's try to cap CPU resources in one zone to demonstrate the basics of resource management in Solaris.

First, run a simple CPU-consuming script in the zone1:

```
root@solaris:~# zlogin zone1 "bash -c 'while true ; do date > /dev/null ; done'"
```

Note that we are simply using `zlogin` to pass the command to the zone.

What's going on in the global zone? Open another window, become root and check:

```
root@solaris:~# vmstat 5
```

Idle is 0, system time is around 70%. Not good.

```
root@solaris:~# zonestat 5
```

Zone1 consumes 70-80% of total resources, the rest is spent in global zone (most likely serving zone1's requests). We decided to reduce the zone1's resource consumption and give it only 50% of our CPU cycles.

```
root@solaris:~# zonecfg -z zone1
zonecfg:zone1> add capped-cpu
zonecfg:zone1:capped-cpu> set ncpus=0.5
zonecfg:zone1:capped-cpu> end
zonecfg:zone1> exit
root@solaris:~# zoneadm -z zone1 reboot
```

After zone1 reboots, log in into it again and repeat the same steps: run the bash script mentioned above and then run `zonestat 5` in another window.

```
root@solaris:~# zlogin zone1 "bash -c 'while true ; do date > /dev/null ; done'"

root@solaris:~# zonestat 5
```

Do you see the difference? Are you happy with the result?

Is it also possible to change this CPU cap parameter on the fly:

```
root@solaris:~# prctl -n zone.cpu-cap -r -v 25 -i zone zone1
```

Check if it works:

```
root@solaris:~# zonestat 5
```

In your virtual appliance there is a simple tool which allows you to visualize the zones workloads. It simply pipes the zonestat output into the popular open source program gnuplot. Try this:

```
root@solaris:~# zoneplot
```

Don't forget to stop the infinite loop in your zone! Or simply halt the zone.

```
root@solaris:~# zoneadm -z zone1 halt
```

Other resources can be capped this way as well: memory, swap, number of threads etc. Again, think about how it can be used in real life situations?

Back to top

Oracle Solaris 11 Security Lab

Table of Contents

Introduction

In this lab we we take quick tour of some of Oracle Solaris security features that can help you to protect your applications and your system. We will look at the following technologies:

- Privileges
- RBAC (Rights and Authorizations)
- Integration with SMF (Service Management Facility)

For lab exercises S.3 and S.4, apache 2.2 is required. Check if it's installed already:

```
root@solaris:~# pkg list *apache*
NAME (PUBLISHER)                                  VERSION                    IFO
web/server/apache-22                              2.2.20-0.175.0.0.0.2.537   i--
```

If Apache web server package is not installed, install it using the following command:

```
root@solaris:~# pkg install apache-22
```

Exercise S.1: Introduction to Solaris RBAC and Privileges

Expected duration: 20 minutes

The goal of this exercise is to gain a basic understanding of the RBAC/privileges framework in Solaris.

Background Information

Thanks to Joerg Moellenkamp and his post on Less known Solaris features: RBAC and Privileges - Part 1: Introduction for the following introduction.

The Story of `root`

And then there was `root`. And `root` was almighty. And that wasn't a good thing. `root` was able to control the world without any control. And `root` needed control. It was only a short chant between the mere mortals and `root`. Everybody with the knowledge of the magic chant was able to speak through `root`.

But `root` wasn't alone. `root` had servants called daemons. Some of one them needed divine powers to do their daily job. But `root` was an undividable being. So the servants had to work with the powers of `root`. But the servants were not as perfect as root: Some of the servants started to do everything mere mortals said to them if they only said more than a certain amount of prayers at once.

Superuser

The old model of rights in a Unix systems is based on a duality. There is the superuser and the normal user. The normal users have a restricted set of rights in the system, the superuser has an unrestricted set of rights. To modify the system, a normal user has to login as `root` directly or assume the rights of root (by `su -`). But such a user has unrestricted access to the system. Often this isn't desirable. Why should you enable an operator to modify a system, when all he or she has do to on the system is create some users from time to time. You've trained them to do `useradd` or `passwd`. What do you do when they get too curious? They need `root` privileges to create a user or change a password. You need some mechanisms to limit this operator.

But it gets more problematic. Programs have to modify the system to work. A web server is a good example. You expect it to use port

80, ports beneath port number 1024 are privileged ports. You need special rights to modify the structures of the system to listen to port 80. A normal user doesn't have these rights. So the web server has to be started as root. The children of this process drop the rights of root by running with a normal user. But there is this single instance of the program with all the rights of the superuser. This process has much more rights than needed, a possible attack vector for malicious users.

Least Privilege

There is a concept in security known as *least privilege*. You give someone only least amount of privileges, only enough to do the tasks they are assigned. An example of the real world - you won't give the janitor the master key for all the rooms on the campus, when all he has to do is work in Building C. The other way around - there are some trusted people who have access to all rooms in case of emergency.

You have the same concept in computer security. Everyone should have only the least amount of privileges in the system to do their job. The concept of the superuser doesn't match to this, it's an all or nothing model. You are either an ordinary user with basic privileges or you are a user with unrestricted rights. There is nothing in between. This doesn't follow the least privileges model.

Role Based Access Control (RBAC)

The example with the key for the janitors is a good example. Let's imagine a large campus. You have janitors responsible for the plumbing (let's call them Lenny and Carl), for the park (let's call him Homer), for the security system (let's call her Marge and Lenny helps her from time to time).

These roles have different sets of privileges: For example the plumbing janitors have access to all the rooms of the heating system. The janitor for the park has only access to the garage with the lawnmower.

When they start to work in their job, they assume a role. From the privilege perspective it's not important who is the person, but what role the person has assumed. Lenny punches the clock and assumes the role of the plumbing janitor for the next 8 hours. And while he is doing its job he uses the privileges inherent to the role. But he also has to do tasks in his office or in his workshop. It's his own room, so he doesn't need the special privileges.

Role Based Access Control is quite similar. You login to the system, and then you start to work. You read your emails (no special privileges needed), you find an email "Create user xy45345. Your Boss". Okay, now you need special privileges. You assume the role of a User Administrator and create the user. Job done, you don´t need the privileges anymore. You leave the role and write the "Job done" mail to your boss with your normal user's privileges.

Role Based Access Control is all about this: Defining roles, giving them privileges and assigning users to this roles.

Privileges

I've used the word quite often in this introduction so far. What is a privilege? A privilege is the right to do something. For example, having the keys for the control panel of the heating system.

Unix users are nothing different. Every user has privileges in a Unix system. A normal user has the privilege to open, close, read, write and delete files when he is allowed to do this (Because he created it, because he belongs to the same group as the creator of the file or the creator gave everybody the right to do it). This looks normal to you, but it's privilege based on the login credentials you gave to system. You don't have the privilege to read all files on the system or to use a port number 1024.

Every thing done in the system is based on these privileges. Solaris has separated the tasks into many privilege sets. At the moment, there are more than 80 different privileges in the system. Normal users have only a basic set of privileges, while `root` has all of them.

In Solaris the model has changed, privileges and users aren't connected with each other. You can give any user the power of the traditional "root user", and restrict the privileges of the root user. Solaris is configured to look as a traditional super user model due to its binary compatibility guarantee that mandates that the standard configuration of the system resembles the superuser model. There are applications out there, which assume that only the `root` user or the UID 0 has unrestricted rights and exit if they are started by a different user.

The only thing that is special with UID 0 in Solaris is that it's the owner of the system configuration files, and thus has the capability to change system configurations.

RBAC and Privileges in Solaris

Both features have their roots in the Trusted Solaris product. Trusted Solaris was a version of Solaris to ensure the highest security standards. Today, these mechanisms are part of the normal Solaris in conjunction with the Trusted Extensions. So RBAC is a really old feature - it has been in Solaris since Solaris 8 (published in 2000). Privileges found their way into the generic Solaris with the first availability of Solaris 10 in February 2005.

Lets start with some basic concepts for RBAC:

- **Rights**: A right is the permission, to execute an executable as an privileged user. For example the permission to execute the command `reboot` as `root`.
- **Authorization**: A permission that enables a user or role to perform a class of actions that could affect security. For example, security policy at installation gives ordinary users the `solaris.device.cdrw authorization`. This authorization enables users to read and write to a CD-ROM device.
- **Right Profiles**: A collection of administrative capabilities that can be assigned to a role or to a user. A rights profile can consist of authorizations, of commands with security attributes, and of other rights profiles. Rights profiles offer a convenient way to group security attributes.
- **Role:** A special identity for running privileged applications. The special identity can be assumed by assigned users only. In a system that is run by roles, superuser is unnecessary. Superuser capabilities are distributed to different roles.

Lets take a closer look at RBAC and roles:

Open a terminal window by right clicking any point in the background of the desktop, and select "Open Terminal" in the pop-up menu.

Lets look at what roles are assigned to our `lab` user. In a terminal, run the following command:

```
lab@solaris:~$ roles
root
```

As you can see here you should be assigned the role `root`, it allows us to assume the `root` role. In Solaris 11 root is not a user but rather a role. So you have to login to the system as a normal user and then assume the `root` role with the `su` command.

In a terminal window type as follows:

```
lab@solaris:~$ userattr type root
role
lab@solaris:~$ auths root
solaris.*
```

As you can see here `root` is defined as a role with all solaris authorizations, the effect of this is similar to the old almighty `root` user. This is put in place for backwards compatibility.

And as your user is assigned the `root` role, you can try this by assuming the `root` role:

```
lab@solaris:~$ whoami
lab
lab@solaris:~$ su
Password: <root password>
lab@solaris:~# whoami
root
lab@solaris:~# who am i
lab   pts/1        Apr 21 07:32 (:0.0)
lab@solaris:~# exit
exit
lab@solaris:~$
```

Lets add a new user, Joe Doe to our system:

```
lab@solaris:~$ useradd -d /export/home/jdoe -m -c "Joe Doe" jdoe
UX: useradd: ERROR: Cannot update system - login cannot be created.
```

As you can see here we are not allowed to use the `useradd` command. We will have to use `su` to execute `useradd` with "root" privileges, and assign missing rights to our account. The `su` command will prompt you for the root password.

```
lab@solaris:~$ su root -c "usermod -K profiles='User Management' lab"
Password: <your root password>

lab@solaris:~$ userattr profiles lab
User Management
```

As you can see, we now have the right to manage users. To take advantage of this right, we will invoke a profile shell, e.g. `pfbash`, and create the new user, jdoe.

```
lab@solaris:~$ pfbash
lab@solaris:~$ useradd  -m -c "Joe Doe" jdoe
```

To change the user's password we have to have User Security profile assigned, but it is not advised to assign it to a regular user because it allows the user to change passwords of other users, including more powerful ones. User Management profile is safe, because you can only delegate what you already have. Therefore, let's change the password of the new user using root role:

```
lab@solaris:~$ su root -c passwd jdoe
Password: <root password>
New Password: abc123
Re-enter new Password: abc123
passwd: password successfully changed for jdoe
lab@solaris:~$ su - jdoe
Password: abc123
Oracle Corporation    SunOS 5.11 11.0    November 2011
jdoe@solaris:~$ roles
No roles
jdoe@solaris:~$ su root
Password: <root password>
Roles can only be assumed by authorized users
su: Sorry
jdoe@solaris:~$ exit
lab@solaris:~$
```

NEW in Role Authentication in Solaris 11

Role Authentication procedure has changed in Solaris 11. Running earlier Solaris version, including Oracle Solaris 11 Express, user had to know role password in order to assume the role. Oracle Solaris 11 includes the ability to specify whether to use the role password or user password when a user wants to assume a given role. Administrators can specify either 'user' or 'role' for the 'roleauth' keyword. If roleauth is not specified, 'role' is implied. Any newly created roles will be 'user' by default.

Summary

As you can see here `jdoe` doesn't have any roles assigned to him and he can't use `su` to assume the `root` role. We'll get back to him later. Now let's move on to a closer look at privileges in Exercise 2.

Exercise S.2: A Closer Look at Solaris Privileges

Expected duration: 20 minutes

The goal of this exercise is to understand Solaris privileges and how to use them.

Background Information

Privileges

What are Privileges? Privileges are rights to do an operation in the kernel. These rights are enforced by the kernel. Whenever you do something within the kernel the access is controlled by privileges.

List of privileges

contract_event, contract_identity, contract_observer, cpc_cpu, dtrace_kernel, dtrace_proc, dtrace_user, file_chown, file_chown_self, file_dac_execute, file_dac_read, file_dac_search, file_dac_write, file_downgrade_sl, file_flag_set, file_link_any, file_owner, file_setid, file_upgrade_sl, graphics_access, graphics_map, ipc_dac_read, ipc_dac_write, ipc_owner, net_access, net_bindmlp, net_icmpaccess, net_mac_aware, net_mac_implicit, net_observability, net_privaddr, net_rawaccess, proc_audit, proc_chroot, proc_clock_highres, proc_exec, proc_fork, proc_info, proc_lock_memory, proc_owner, proc_priocntl, proc_session, proc_setid, proc_taskid, proc_zone, sys_acct, sys_admin, sys_audit, sys_config, sys_devices, sys_ipc_config, sys_linkdir, sys_mount, sys_iptun_config, sys_dl_config, sys_ip_config, sys_net_config, sys_nfs, sys_ppp_config, sys_res_config, sys_resource, sys_smb, sys_suser_compat, sys_time, sys_trans_label, virt_manage, win_colormap, win_config, win_dac_read, win_dac_write, win_devices, win_dga, win_downgrade_sl, win_fontpath, win_mac_read, win_mac_write, win_selection, win_upgrade_sl, xvm_control

Legend: blue - basic privilege

Conventional Unix

On conventional Unix systems you have a `root` user and `root` has all privileges. And you have a normal user, who has only a limited set of privileges. Sometimes you need the rights of an admin to do some tasks. You don't even need to administer the system.

```
lab@solaris:~$ ls -l /usr/sbin/traceroute
-r-sr-xr-x 1 root bin 46868 2010-11-05 08:02 /usr/sbin/traceroute
lab@solaris:~$ ls -l /usr/sbin/ping
-r-sr-xr-x 1 root bin 55940 2010-11-05 08:01 /usr/sbin/ping
```

You can use `traceroute` and `ping` because both tools are `setuid` tools. `setuid` allows a process to run with a specific userid and **all** the privileges that come with that user. In the case of `setuid` to root user the process would have root's privileges. A `setuid` program in Solaris can be privilege-aware and hence only retain the privileges needed for the operation.

You need a special privilege to `ping` - the privilege to use access ICMP. On conventional systems this right is reserved for the `root` user. Thus the `ping` program has to be executed with the rights of `root`. The problem: At the time of the execution of the program, the program has all the rights of the user. Not only to access ICMP, but the program is capable of doing everything on the system, such as deleting files in `/etc`. This may not a problem with `ping` or `traceroute` but think about larger programs. An exploit in a `setuid` program can lead to the escalation of the user's privileges.

Let's have a look at the privileges of an ordinary user. There is a tool to get the privileges of any given process in the system, it's called `ppriv`. The variable `$$` is a shortcut for the actual process id (in this case the process id of the user's shell):

```
lab@solaris:~$ ppriv $$
1684:    -bash
flags = <none>
    E: basic
    I: basic
    P: basic
    L: all
```

To find out what "basic" means, we can add `-v` to the `ppriv` flag to expand the aliases:

```
lab@solaris:~$ ppriv -v $$
1684:    -bash flags = <none>
        E: file_link_any,file_read,file_write,net_access,proc_exec,proc_fork,proc_info,proc_sess
        I: file_link_any,file_read,file_write,net_access,proc_exec,proc_fork,proc_info,proc_sess
        P: file_link_any,file_read,file_write,net_access,proc_exec,proc_fork,proc_info,proc_sess
        L: contract_event,contract_identity,contract_observer,cpc_cpu,dtrace_kernel,dtrace_proc,
```

Every process in the system has four sets of privileges that determine if a process is enabled to use a privilege or not. The theory of privileges is quite complex. I would suggest to read the chapter How Privileges Are Implemented" in the Developers Guide to Oracle Solaris Security to learn how each set controls or is controlled other privilege sets.

At this time, I want only to explain the meaning of the first letter:

- **E**: **E**ffective privileges set
- **I**: **I**nheritable privileges set
- **P**: **P**ermitted privileges set
- **L**: **L**imit privileges set

You can think about the privilege sets as keyrings. The *effective* privilege set are the keys the janitor has on its keyring. The *permitted* privilege set are the keys the janitor is allowed to put on its keyring. The janitor can decide to remove some of the keys. Perhaps he thinks: I work only in room 232 today, I don't need all the other keys. I'll leave them in my office. When he looses his keyring he lost only the control of this single room, not about the complete campus.

The *inheritable* privilege set is not really a keyring. The janitor thinks about his new assistant: "Good worker, but I will not give him my key for the room with the expensive tools." The *limited privilege* set is the overarching order from the boss of janitor to his team leaders: "You are allowed to give your assistant the keys for normal rooms, but not for the rooms with all this blinking boxes from Oracle".

At the moment the most interesting set is the E:. This is the effective set of privileges. This is the set of privilege effectively available to process. Compared to the full list of privileges mentioned above the set is much smaller. But this matches your experience when you use a Unix system.

Lets see how it looks when we assume the root role:

```
lab@solaris:~$ su -
Password: <root password>
Oracle Corporation    SunOS 5.11    11.0    November 2011
You have new mail.
root@solaris:~# ppriv $$
1779:    -bash
flags = <none>
    E: all
    I: basic
    P: all
    L: all
```

This role has much more privileges. The effective set is much broader. When we assume the root role we assume all privileges in the system. NOTE: This is strictly to provide backwards compatibility and is not the recommended usage going forward.

Lets add some privileges to our `jdoe` user, let's assume that he is a software developer that needs access to DTrace to debug some applications.

First lets try to use DTrace as jdoe:

```
# su - jdoe
Password: abc123
Oracle Corporation    SunOS 5.11    11.0    November 2011
jdoe@solaris:~$ ppriv $$
1790:    -bash
flags = <none>
    E: basic
    I: basic
    P: basic
    L: all
jdoe@solaris:~$ dtrace -l
dtrace: failed to initialize dtrace: DTrace requires additional privileges
```

As we can see here `jdoe` is missing some privileges to be allowed to use DTrace. There are have 3 privileges needed for DTrace: `dtrace_kernel, dtrace_proc, dtrace_user`, depending on what the user needs to use DTrace for. We can add these privileges to the user in three different ways:

1. Add the privileges needed to the user. This would allow the user to use DTrace as him or herself.
2. Create an execution profile with the `dtrace` command and the needed privileges, and assign the profile to the user. This would allow him or her to use a profile shell (i.e., `pfbash`) to execute the `dtrace` command.

3. Create a role with the needed privileges, and assign the role to user. This would require the user to assume the role to use DTrace.

Lets see how to add dtrace privileges directly to the user:

```
root@solaris:~# usermod -K defaultpriv=basic,dtrace_kernel,dtrace_proc,dtrace_user jdoe
```

Let's see if `jdoe` now has all needed privileges to use the `dtrace` command:

```
lab@solaris:~$ su - jdoe
Password: abc123
Oracle Corporation    SunOS 5.11    11.0    November 2011
jdoe@solaris:~$ ppriv $$
1814: -bash
flags = <none>
    E: basic,dtrace_kernel,dtrace_proc,dtrace_user
    I: basic,dtrace_kernel,dtrace_proc,dtrace_user
    P: basic,dtrace_kernel,dtrace_proc,dtrace_user
    L: all
jdoe@solaris:~$ dtrace -l | more
   ID   PROVIDER            MODULE                          FUNCTION NAME
    1     dtrace                                                     BEGIN
    2     dtrace                                                     END
    3     dtrace                                                     ERROR
    7    syscall                                                  nosys entry
    8    syscall                                                  nosys return
    9    syscall                                                  rexit entry
   10    syscall                                                  rexit return
```

OK, so that worked, just for the fun of it lets try to create a role that does the same.

First lets remove the dtrace privileges from jdoe:

```
root@solaris:~# usermod -K defaultpriv=basic jdoe
root@solaris:~# userattr defaultpriv jdoe
basic
```

Then lets create a new role (using root shell, i.e. as a *root* user), lets call the role "bugger", and assign it to `jdoe`

root@solaris:~# `roleadd -u 201 -d /export/home/bugger -P "Process Management" bugger` root@solaris:~# `passwd bugger` New Password: `abc123` Re-enter new Password: `abc123` passwd: password successfully changed for bugger root@solaris:~# `rolemod -K defaultpriv=basic,dtrace_kernel,dtrace_proc,dtrace_user bugger` root@solaris:~# `userattr defaultpriv bugger` basic,dtrace_kernel,dtrace_proc,dtrace_user root@solaris:~# `usermod -R bugger jdoe` root@solaris:~# `roles jdoe` bugger

OK, lets try and see if `jdoe` now can use DTrace

```
lab@solaris:~$ su - jdoe
Password: Oracle Corporation    SunOS 5.11    11.0    November 2011
jdoe@solaris:~$ roles
bugger
jdoe@solaris:~$ su bugger
Password:
jdoe@solaris:~$ ppriv $$
1855:   bash
flags = PRIV_PFEXEC
        E: basic,dtrace_kernel,dtrace_proc,dtrace_user
        I: basic,dtrace_kernel,dtrace_proc,dtrace_user
        P: basic,dtrace_kernel,dtrace_proc,dtrace_user
```

```
        L: all

jdoe@solaris:~$ dtrace -l | more
   ID   PROVIDER            MODULE                          FUNCTION NAME
    1     dtrace                                                     BEGIN
    2     dtrace                                                     END
    3     dtrace                                                     ERROR
    7    syscall                                                  nosys entry
    8    syscall                                                  nosys return
    9    syscall                                                  rexit entry
   10    syscall                                                  rexit return
```

Summary

Now you should have a basic understanding about privileges and how you can assign them to users, however wouldn't it be nice if we could do the same to processes? We will look at that in Exercise S.3.

Exercise S.3: Process Privileges and Rights

Expected duration: 10 minutes

Background

We have looked at how you can add privileges to a user, now we'll look at how privileges interact with processes, we will look at both processes that are *privilege aware* and processes that are *non-privilege aware*.

The idea of managing privileges is not limited to users and their shells. In any given system you find dozens of programs running as daemons.

These daemons interact in several ways with the privileges. The best way is "*privilege-aware programming*". Let's assume you code a daemon for your system. For example: You know that your daemon will never do an `exec()` call. So you can safely drop this privilege. The process modifies the permitted privilege set. The process can remove a privilege but not add it. Even when someone is able to access your process, the attacker can't make an `exec()` call. The process doesn't even have the privilege to do such a call. And the attacker can't add the privilege again. Several processes and programs in Solaris are already privilege aware. For example the kernel-level cryptographic framework daemon. Let's look at the privileges of the daemon.

In order to observe or manage privileges, a process must itself already have those privileges. Therefore we will assume the root role for the following exercise.

```
lab@solaris:~$ su - root
Password: <root password>
root@solaris:~# ps -ef | grep kcfd
  daemon    80     1   0 06:04:21 ?           0:00 /lib/crypto/kcfd

root@solaris:~# ppriv -v 80
80:     /lib/crypto/kcfd
flags = PRIV_AWARE
        E: file_owner,file_read,file_write,net_access,proc_priocntl,sys_devices
        I: none
        P: file_owner,file_read,file_write,net_access,proc_priocntl,sys_devices
        L: none
```

This daemon doesn't have even the basic privileges of a regular user. It has the only the bare minimum of privileges to do its job.

Non-privilege aware processes

But the world isn't perfect. Not every process is privilege aware. Thus you have to limit the privileges by other mechanisms. The Service Management Facility comes to help. The following example is copied from Glenn Brunette's Blueprint Limiting Service Privileges in the Solaris 10 Operating System.

Let's take the Apache Webserver as an example. The apache web server is not privilege aware. We start the daemon via the Service Management Framework (SMF).

```
root@solaris:~# svcadm -v enable -s apache22
svc:/network/http:apache22 enabled.
```

OK, now we look at the processes of the Apache daemons.

```
root@solaris:~# ps -ef | grep "apache" | grep -v grep
webservd  1894  1891   0 12:35:00 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
webservd  1892  1891   0 12:35:00 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
webservd  1893  1891   0 12:35:00 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
    root  1891     1   0 12:34:59 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
webservd  1895  1891   0 12:35:00 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
webservd  1896  1891   0 12:35:00 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
```

Five daemons are running as webservd, and one is running as root.

```
root@solaris:~# ppriv 1891
1891:   /usr/apache2/2.2/bin/httpd -k start
flags =
        E: all
        I: basic
        P: all
        L: all
```

As expected for a root process, this process has the complete set of privileges of a root user. Now lets look at one of its children:

```
root@solaris:~# ppriv 1895
1895:   /usr/apache2/2.2/bin/httpd -k start
flags =
        E: basic
        I: basic
        P: basic
        L: all
```

Much better... Only basic privileges.

OK, There is a reason for this configuration. On Unix systems, you have two groups of ports. Privileged ones from 1-1023 and unprivileged ones from 1024 up. You can only bind to a privileged port with the privilege to do it. A normal user doesn't have this privilege, but `root` does. And thus there has to be one process running as root. Do you remember the list of privileges for the apache process running at root? The process has all privileges but needs only one of them, that isn't part of the basic privilege set.

Exercise S.4: SMF, Process Privileges and Authorizations

Expected duration: 10 minutes

Introduction

We have now looked at both privilege aware and non-privilege aware processes, now lets look at how we can use SMF to let a user with a limited set of privileges manage a non-privilege aware process and give it the required privileges without using setuid to root.

In order to observe or manage privileges, a process must itself already have those privileges. Therefore we will assume the root role for the following exercise.

Background

How to get rid of the `root` apache process

It doesn't have to be this way. With Solaris you can give any user or process the privilege to use a privileged port. You don't need the

root process anymore.

Let's configure it this way. At first we have to deactivate the running apache from Exercise S.3:

```
lab@solaris:~$ su - root
Password: <root password>
root@solaris:~# svcadm -v disable -s apache22
svc:/network/http:apache22 disabled.
```

We will not be explaining the Service Management Facility here, but you can set certain properties in SMF to control the startup of a service:

```
1  root@solaris:~# svccfg -s apache22
2  svc:/network/http:apache22> setprop start/user = astring: webservd
3  svc:/network/http:apache22> setprop start/group = astring: webservd
4  svc:/network/http:apache22> setprop start/privileges = astring: basic,!proc_session,!proc_info,!fil
5  svc:/network/http:apache22> setprop start/limit_privileges = astring: :default
6  svc:/network/http:apache22> setprop start/use_profile = boolean: false
7  svc:/network/http:apache22> setprop start/supp_groups = astring: :default
8  svc:/network/http:apache22> setprop start/working_directory = astring: :default
9  svc:/network/http:apache22> setprop start/project = astring: :default
10 svc:/network/http:apache22> setprop start/resource_pool = astring: :default
11 svc:/network/http:apache22> end
```

Line 2 to 4 are the most interesting ones. Without any changes, the Apache daemon starts as root and forks away processes with the webservd user. But we want to get rid of the root user for this configuration. Thus we start the daemon directly with the webservd user, the same goes for the group id.

Now it gets interesting. Without this line (line 4), the kernel would deny Apache to bind to port 80. webservd is a regular user without the privilege to use a privileged port. The property `start/privileges` sets the privileges to start the service. At first, we give the service basic privileges. Then we add the privilege to use a privileged port. The service would start up now.

But wait, we can do more. A webserver shouldn't do any hardlinks. And it doesn't send signals outside it's session. And it doesn't look at processes other than those to which it can send signals. We don't need these privileges. **proc_session**, **proc_info** and **file_link_any** are part of the basic privilege set. We remove them, by adding a ! in front of the privilege:

```
root@solaris:~# svcadm -v refresh apache22
Action refresh set for svc:/network/http:apache22.
```

OK, we have notified SMF of the configuration changes.

Until now, the apache daemon used the `root` privileges. Thus the ownership of files and directories were unproblematic. The daemon was able to read and write in any directory or file in the system. As we drop its privilege set by using a regular user, we have to modify the ownership of some files and move some files:

```
root@solaris:~# mkdir -p -m 755 /var/apache2/2.2/run
root@solaris:~# chown webservd:webservd /var/apache2/2.2/run
root@solaris:~# chown webservd:webservd /var/apache2/2.2/logs/access_log
root@solaris:~# chown webservd:webservd /var/apache2/2.2/logs/error_log
```

We need some configuration changes, too. We have to move the LockFile and the PidFile. There wasn't one of the two configuration directives specified earlier, thus we're simply appending them to the end of the configuration file:

```
root@solaris:~# bash -c 'echo LockFile /var/apache2/2.2/logs/accept.lock >> /etc/apache2/2.2/httpd.conf
root@solaris:~# bash -c 'echo PidFile /var/apache2/2.2/run/httpd.pid >> /etc/apache2/2.2/httpd.conf'
```

OK, everything is in place. Let´s give it a try.

```
root@solaris:~# svcadm disable apache22
```

```
svc:/network/http:apache22 disabled.
root@solaris:~# svcadm -v enable -s apache22
svc:/network/http:apache22 enabled.
```

Now we check for the running httpd processes:

```
root@solaris:~# ps -ef | grep httpd | grep -v grep
webservd  2032  2031   0 14:34:27 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
webservd  2031     1   1 14:34:26 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
webservd  2033  2031   0 14:34:27 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
webservd  2034  2031   0 14:34:27 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
webservd  2036  2031   0 14:34:27 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
webservd  2035  2031   0 14:34:27 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
```

You notice the difference ? There is no httpd running as root. All processes run with the userid webservd. Mission accomplished.

Let's check the privileges of the processes. At first the one, who ran as `root` before:

```
root@solaris:~# ppriv 2031
2031:   /usr/apache2/2.2/bin/httpd -k start
flags = <none>
        E: basic,!file_link_any,net_privaddr,!proc_info,!proc_session
        I: basic,!file_link_any,net_privaddr,!proc_info,!proc_session
        P: basic,!file_link_any,net_privaddr,!proc_info,!proc_session
        L: all
```

Only the least privileges to do the job, no root privileges. And even the other processes are more secure now. Before we changed the configuration of the webserver, it had the basic privileges of a regular user. Now we have limited even this set.

But we still need to be a privileged user to start apache. Wouldn't it be nice if I could delegate management of "services" to unprivileged users. I can use authorizations to do that, and authorizations is integrated with SMF. There are two authorizations we will need to add to our user to give hi the right to manage the apach22 service. We need an authorization to manage our service as well as a authorization to change the state of our service:

- `solaris.smf.value.http/apache22` - Gives a user the right to modify the value of a property in the Apache service.
- `solaris.smf.manage.http/apache22` - Gives a user the right to change the state of a the Apache service with `svcadm`.

Both are needed to change the state of a service, as the state of the service is kept as a property for the service.

```
root@solaris:~# svcprop http:apache22 | grep auth
httpd/value_authorization astring solaris.smf.value.http/apache22
general/action_authorization astring solaris.smf.manage.http/apache22
general/value_authorization astring solaris.smf.value.http/apache22
```

First we need to add our authorizations to `/etc/security/auth_attr`

```
root@solaris:~# echo solaris.smf.manage.http:::Mange Apache 2.2:: >> /etc/security/auth_attr
root@solaris:~# echo solaris.smf.value.http:::Mange Apache 2.2:: >> /etc/security/auth_attr
```

Note: In most SMF manifests one authorization is used for both action_authorization and value_authorization, as they are commonly used together.

Now lets add those authorizations to our old friend `jdoe`.

```
root@solaris:~# usermod -A solaris.smf.manage.http/apache22,solaris.smf.value.http/apache22 jdoe
Found user in files repository.
root@solaris:~# userattr auths jdoe
solaris.smf.manage.http/apache22,solaris.smf.value.http/apache22
```

Lets see if `jdoe` can start and stop apache now

```
root@solaris:~# su - jdoe
Oracle Corporation      SunOS 5.11      11.0     November 2011
jdoe@solaris:~$ svcadm disable apache22
jdoe@solaris:~$ svcs *apache*
STATE          STIME    FMRI
disabled       14:57:51 svc:/network/http:apache22
jdoe@solaris:~$ svcadm enable apache22
jdoe@solaris:~$ svcs *apache*
STATE          STIME    FMRI
online         14:58:24 svc:/network/http:apache22
```

That worked fine, so by using SMF we've been able to take a service such as apache, remove its need to start as root by giving it the privileges it needed and nothing more. We also removed some privileges that apache doesn't need, but a hacker might use them to take control of your system.

Add to that we added the right to manage the apache service via SMF to an unprivileged user by giving him the correct authorizations that allows a user to manage that particular service and nothing else, as you can see here.

```
jdoe@solaris:~$ svcs sendmail
STATE          STIME    FMRI
online          6:48:12 svc:/network/smtp:sendmail
jdoe@solaris:~$ svcadm disable sendmail
svcadm: svc:/network/smtp:sendmail: Permission denied.
```

Congratulations! You have successfully completed Oracle Solaris 11 Security Lab.

For additional information about the technologies used in this lab, please see the following links:

http://www.oracle.com/technetwork/server-storage/solaris/overview/security-163473.html
http://www.oracle.com/us/products/servers-storage/solaris/security/index.html

Restricting Service Administration in the Solaris 10 OS
Limiting Service Privileges in the Solaris 10 OS
Privilege Debugging in the Solaris 10 Operating System
Privilege Bracketing in the Solaris 10 Operating System

Oracle Solaris 11 DTrace Lab

Table of Contents

Exercise D.1: DTrace CPU

Task: You have noticed that system utilization is very high. How to find the process which consumes most of the resources?

Lab: Let's model a simple situation to demonstrate how DTrace scripts can be used in the situations when no other tool is helpful.

60 Seconds of Theory: DTrace is a big topic. We can spend weeks and months learning DTrace and we will still be discovering something new. The "DTrace Book" is hot off the press and available at Amazon. It's over 1150 pages on the subject! We'll spend about half an hour just to give you a taste of DTrace. If you need more-buy the book, find DTrace resources on the web--there are plenty of them.

In short, DTrace is a tool which allows you to look into every part of your system: from kernel structures and system calls to application functions, from contents of system stack to user interface. It is achieved by embedding a lot (literally, tens of thousands!) of probes in each and every place of the operating system. All those probes are dynamic: you can turn them on and off (fire them) and they don't consume system resources when not in use.

Another important feature of DTrace is its safety. Safety was one the central design goals of DTrace from its inception; that means you can (and should) use DTrace on your production systems.

DTrace uses a C-like scripting language called D, which supports all ANSI C operators. Most likely you are familiar with C language syntax, that really helps in learning D. We won't go into DTrace language and syntax here, we'll just try several examples that can be helpful in everyday life.

DTrace one-liners

There are a lot of useful one-liners in every scripting language: bash, sed, awk, Perl etc. DTrace is not an exception here. Try the following one-liners and see what you can observe with them. The usual way to work with DTrace scripts is to start them, wait for some time while DTrace is collecting data and then press Ctrl-C.

What processes are being started currently? New processes with arguments:

```
dtrace -n 'proc:::exec-success { trace(curpsinfo->pr_psargs); }'
```

Longer version: new processes with arguments and time

```
dtrace -qn 'syscall::exec*:return { printf("%Y %s\n",walltimestamp,curpsinfo->pr_psargs); }'
```

Which files are being opened by each starting process?

```
dtrace -n 'syscall::open*:entry { printf("%s %s",execname,copyinstr(arg1)); }'
```

System time is high. What are the programs doing, who is calling most of system calls?

```
dtrace -n 'syscall:::entry { @num[execname] = count(); }'
```

Which syscalls are being called most often?

```
dtrace -n 'syscall:::entry { @num[probefunc] = count(); }'
```

Which process is calling most of system calls?

```
dtrace -n 'syscall:::entry { @num[pid,execname] = count(); }'
```

Read IOPS are high. Who is reading now? Read bytes by process

```
dtrace -n 'sysinfo:::readch { @bytes[execname] = sum(arg0); }'
```

Write IOPS are high. Somebody is writing to disk, filling up all the space. Who is writing?

```
dtrace -n 'sysinfo:::writech { @bytes[execname] = sum(arg0); }'
```

How big are blocks being read? Read size distribution by process

```
dtrace -n 'sysinfo:::readch { @dist[execname] = quantize(arg0); }'
```

Write size distribution by process

```
dtrace -n 'sysinfo:::writech { @dist[execname] = quantize(arg0); }'
```

Back to our example. Open another terminal window or a tab on your Solaris desktop. Write the following bash script:

```
lab@solaris:~$ while true ; do date > /dev/null ; done
```

In your previous window/tab (where you have your root session open) run:

```
root@solaris:~# vmstat 1
```

What do you see? System time as about 70%, user time is about 30%, idle is 0%. System is really busy doing something in the kernel. Imagine you don't know what has happened in the other window. What would you do?

```
root@solaris:~# top
```

...shows nothing. Not a single process in the list is generating such high load.

```
root@solaris:~# prstat
```

...doesn't help as well. OK, let's think: high system time means a lot of system calls. Use DTrace and ask: which program generates the most of the system calls?

```
root@solaris:~# dtrace -n 'syscall:::entry { @num[execname] = count(); }'
```

Does it look cryptic to you? No worries: we have just instructed DTrace to count ("`count()`") every system call ("`syscall`") when it starts ("`entry`") then aggregate that numbers of system calls by program name ("`execname`") and sort the output.

Wait a little bit (DTrace is collecting the data) and press Ctrl-C. You see: at the bottom of the list there is the "date" command. That means "date" is issuing a lot of system calls. We can try to find a program with the name "date":

```
root@solaris:~# ps -ef | grep date
```

Nothing. Let's try another script: what new processes are being executed?

```
root@solaris:~# dtrace -qn 'syscall::exec*:return \
{ printf("%Y %s\n",walltimestamp,curpsinfo->pr_psargs); }'
```

Now we are counting not every system call, but just exec* system calls. Then we instruct DTrace to output time and process arguments.

A-ha, now we see: a lot of "date" commands are executed each second! That's why we didn't see them in `ps -ef`! They just start and finish in a matter of milliseconds. But who runs all these "date" commands? Let's modify the last script a little bit and make it print out not only the arguments, but also the process ID and the parent process ID:

```
root@solaris:~# dtrace -qn 'syscall::exec*:return \
{ printf("%Y %s %d %d\n",walltimestamp,curpsinfo->pr_psargs,curpsinfo->pr_pid,curpsinfo->pr_ppid); }'
```

We see a lot of strings like this:

```
2011 Aug 5 20:32:24 date 28996 29016
2011 Aug 5 20:32:24 date 28997 29016
```

Now it's clear that a lot of "date" calls are generated by the process ID 29016! What is that process and who has started it?

```
root@solaris:~# ps -f -p 29016
```

```
UID PID PPID C STIME TTY TIME CMD
lab 29016 1608 12 16:27:26 pts/4 2:09 bash
```

OK, now we know who's to blame!

Don't forget to kill the shell process which runs the infinite loop!

Back to top

Exercise D.2: DTrace Disk

Task: You have noticed that free disk space has decreased dramatically and keeps decreasing very fast. Who is "eating" our disk space?

In this lab we are going to use DTrace Toolkit script. More about DTrace Toolkit in the tip below. You don't have to run all those scripts during the lab, leave it for your homework exercise.

DTrace Toolkit

Another step in learning DTrace is the excellent Toolkit written by Brendan Gregg:

`http://www.brendangregg.com/dtrace.html#DTraceToolkit`

The toolkit includes more than 200 very useful, well documented scripts for system administrator's everyday use. Here is a short description of it's content:

`http://www.solarisinternals.com/wiki/index.php/DTraceToolkit`

The good news is that it's already installed in the default Solaris 11 installation and it's located here: /usr/dtrace/DTT/

It is highly recommended to start with the most useful scripts which are located in main DTT directory and are executable from command line.

Quote from `http://www.solarisinternals.com/wiki/index.php/DTraceToolkit#Top_Scripts_to_Run` :

If you have the DTraceToolkit on a misbehaving server and you don't know where to start, the following list of tools will provide the most valuable info in the shortest time:

1. `./execsnoop -v` : Look for many processes being executed quickly, as many short lived processes add considerable overhead.

2. `./iosnoop` : Watch what is happening on the disks for any unexpected activity. Check who is using the disks and the size of the disk events.

3. `./opensnoop -e` : Run `opensnoop`, learn lots. It is amazing what interesting problems `opensnoop` has unearthed. Many things are files (config files, data files, device files, libraries), and watching the open events provides a useful insight on what is really happening.

4. `./procsystime -aT` : This will show elapsed time, on-CPU time and counts for system calls. Very useful, and either -p PID or -n execname can be used to narrow examination to your target application only.

5. `./iotop -PCt8` : If disk events occured too quickly, iotop can provide a rolling summary.

6. `./rwtop -Ct8` : Rather than looking at the disk event level (`iosnoop`, `iotop`), `rwtop` examines at the syscall layer. This is application I/O. Much of this may be to the file system cache, some may make it to disk, and some may be for IPC.

Also don't forget to have some fun with the scripts from Misc/ directory!

Lab: To simulate this problem, we are going to use a small program that does nothing but just eats disk space. First argument is the rate (in kB/sec) at which it will be eating space; second argument is a name of big file we are going to create for that. (Just in case: the program is located in `/home/lab/bin/.`)

```
root@solaris:~# diskeater.py -s 100 -o /export/home/lab/bigfile
```

Open another terminal window, become a "`root`" there and check free space in your system's `/export/home` directory:

```
root@solaris:~# df -b /export/home
```

Repeat this several times and take a note how your free space is decreasing. You can even try the following script to automate it:

```
root@solaris:~# while true ; do df -b /export/home ; sleep 1 ; done
```

How can we find out which process is eating our disk space? Usually programs like `iostat` don't give you per-process information, only per-disk I/O workload. Try this:

```
root@solaris:~# iostat -x 5
```

What do you see? Some writing activity on one of the disks, but nothing more detailed. Think about other ways to solve this puzzle. Here is how it can be solved with DTrace Toolkit:

```
root@solaris:~# /usr/dtrace/DTT/rwtop
```

You will see something like this:

```
2011 Nov 15 13:11:39,  load: 0.25,  app_r:      2 KB,  app_w:    502 KB

  UID    PID   PPID CMD              D            BYTES
60004   1710   1641 nautilus         R                0
60004   1713   1641 updatemanagernot R                0
60004   1722   1641 isapython2.6     R                0
60004   1751   1641 gnome-power-mana R                0
60004   1762   1641 nwam-manager     R                0
60004   1735   1641 java             R                1
60004   1763   1641 xscreensaver     W                8
60004   1792      1 gnome-terminal   R               25
60004    810    782 Xorg             W               32
60004   1763   1641 xscreensaver     R               32
60004   1735   1641 java             W               33
60004   1792      1 gnome-terminal   W             2492
60004    810    782 Xorg             R             2532
60004   1811   1810 diskeater        W           512000
```

Now it's pretty easy to identify who is responsible for our free space deficit!

Don't forget to stop the `diskeater` process in the other window, otherwise the problem will become very real (at least inside your virtual machine).

Back to top