Global Information Assurance Certification Paper

Veritas Volume Manager and a Storage Area Network

Joseph Bell
January 19, 2005

Abstract

This paper will discuss the task of installing, configuring and securing Veritas Volume Manager (VxVM). VxVM is an "advanced, system-level disk and storage array solution that alleviates downtime during system maintenance by enabling easy, online disk administration and configuration. The product also helps ensure data integrity and high availability by offering fast failure recovery and fault tolerant features."[1] Additionally, I will discuss incorporating the VxVM resources into an existing Veritas Cluster Server (VCS) configuration. The purpose of this paper is not only to walk the reader through what I've presented above, but to also provide information that I wasn't able to find in a text or on the Internet and was only able to learn by way of trial and error. I've also supplied the Perl code I wrote to perform the installing, patching and configuring of VxVM.

[1] Quote taken from http://www.cuddletech.com/veritas/vxcrashkourse/vxcrashkourse.pdf page 3.

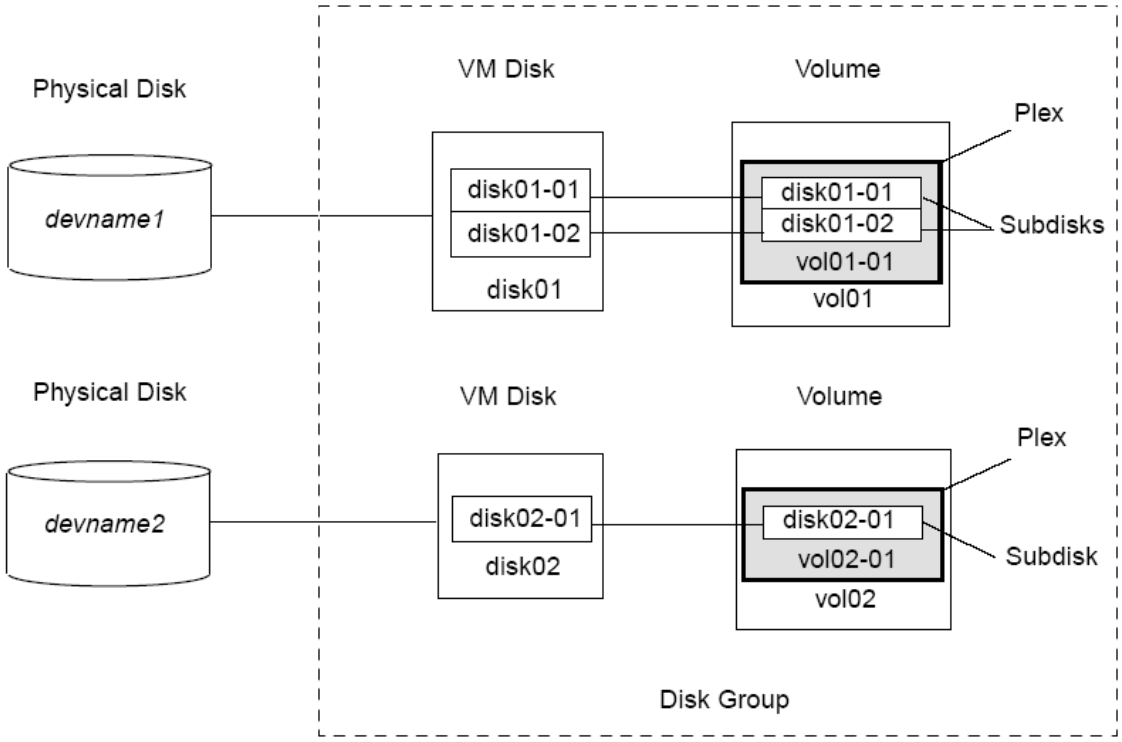
Before Snapshot

I'll provide some background information by defining and describing the tools used and the hardware configuration of the system utilized. Then, I will describe my task and give the reader an idea of my starting conditions and knowledge.

Veritas Volume Manager™

According to the VxVM 3.5 Admin Guide (1), VxVM is defined as "a storage management subsystem that allows you to manage physical disks as logical devices called *volumes*." The key word in that definition is "logical" and the reference to a volume requires a considerable definition in itself. A volume is a logical device, which is composed of one or more plexes. A plex is another logical device and is most commonly referred to as a mirror. Two plexes that are contained within the same volume are mirrors of one another. However many plexes you have within a volume is up to you. The more plexes you have simply increases availability of the data they contain and reduces the chance that you will loose that data, but at the cost of increased storage. A plex is composed of one or more subdisks, another logical device. A subdisk resides on a VM disk, another logical object. One or more subdisks can be associated with a single VM disk. A VM disk is composed of a "real" device as per the operating system's view. I quote the term real because the device the operating system sees may be a local hard disk or it may be a volume on a storage area network, which is managed by another product entirely. The later case is my situation. I have a storage area network managed by a product called SANtricity. A thorough definition of SANtricity is outside the scope of this paper. I will simply define it as a COTS product that manages RAID volumes. SANtricity is offering up multiple volumes to the operating system with each volume appearing as a single device. A collection of VM disks with a common configuration is called a disk group, which is a logical device. It is this device that allows for a group of disks to be utilized as a single entity and allows a user or application (VCS) to move the disk group and its components from one host to another. To recap, a disk group is actually at the top of the VxVM logical food chain. It is composed of one or more VxVM volumes; not to be confused with the volumes offered up by SANtricity. Below is an illustration of the logical devices previously described.

Connection Between Objects in VxVM



[2]

Dirty Region Logging (DRL) is VxVM's fault recovery mechanism, which you won't really appreciate unless you are mirroring large volumes and your system crashes. The reason for this is that the larger the volume, the longer it takes to re-mirror. In short, DRL keeps a log of what has changed across an entire volume all within a bitmap. The bitmap represents the entire volume with each bit corresponding to a region of the volume. Before a region of the disk is written to, the corresponding bit is marked dirty. So, if your system crashes and you have DRL enabled, the time to recover the mirrored volume can be considerably shorter than with it not enabled because only the regions marked dirty need to be accessed. This can greatly reduce the time required to get a system back to an operational state in the event that something causes your system to panic and forces a reboot.

Rebooting is another issue of note. I made the mistake of issuing a reboot

[2] Figure taken from Veritas Volume Manger™ 3.5 Administrator's Guide, pg 13.

command and learned the hard way that VxVM much prefers an 'init 6' call. A reboot doesn't allow the shutdown scripts in the /etc/rcX.d directories to executed. It is those scripts that shut down VxVM and its resources gracefully. This allows for those resources to come up as expected on reboot. This is similar to the operating system syncing the file systems on reboot.

Storage Area Network

A storage area network (SAN), as a subject, is far too vast to fully address within this paper. A semi-formal definition of a SAN is, "a storage area network (SAN) can address several challenges faced by system administrators. Unlike direct-attached storage (DAS), sans allow the administrator to manage a central pool of storage and allocate it to individual hosts as needed. Furthermore, the optical nature of SANs provide flexibility not available with direct-attached storage which typically uses electrical signaling" (2). My need of a SAN is most easily defined as a central repository for data in a networked environment with a need for fibre optical connections due to the file sizes being utilized. I need a SAN to place files for a short period of time. Other servers on the network will retrieve those files when their resources permit them to. The actual time these files will be on the SAN is determined by those servers receiving them and can take a considerable amount of time to process as the files are on average 50 gigabytes in size. The total space on my SAN is a little more than 4 terabytes, which may sound like a lot of space unless you are dealing with 50 GB files.

Veritas Cluster Server™

VCS is a application used primarily to create and manage nodes (servers) within a clustered network and the applications that run on those servers. The true benefit of VCS is found in its ability to perform what is called failover. Failover is best described as the process of bringing a resource down on one server and then bringing that resource up on another server with no user interaction required. This functionality is particularly useful when an application is under VCS control and that application fails on a particular server. VCS will sense this failure and can attempt to bring the resource back up on the same server or on another server depending on how you have it configured to handle failure. An administrator places the resources they want managed by VCS under VCS' control. Each resource type has an agent that is responsible for managing and monitoring that resource. A few examples of resource types are mount, disk, disk group, and share.

My Scenario

I have two SunFire 6800 servers with read-write access to a SAN with ten

clients. Each client has read-only access. The two 6800's are called srvr-A and srvr-B. I have a set of disks that are to be used exclusively as metadata disks. Metadata is basically data about data, which in this case contains the classification of the file on a scale of 1-10 and the size of the file. While the operating system does maintain the size of the file within that file's inode, I need it kept separately as part of the metadata so that the software transferring the actual data file can verify that those two numbers stay consistent as the file propagates through the system. Doing this allows the integrity of the file to be verified at multiple places as the file propagates through the system and allows for alarms to be generated in the event that the file becomes corrupted. I need the data on those disks mirrored, which is where my need of VxVM comes in. It should be noted here that there are multiple products that will provide this functionality and do so much more cost effectively, but using VxVM is an actual program requirement. It's overkill to use VxVM in just this manner as it does so much more, but I'm just the engineer. What do I know? Additionally, the licenses I've been supplied with did not come with the cluster support feature, which means I can't use these metadata disks as shared storage between srvr-A and srvr-B. Again, I'm confused. Why buy a product as expensive as VxVM, require so little of its functionality and then take the cheap route on the licenses? Not my decision to make and my opinion wasn't requested, but a point I would certainly have tried to make if given the opportunity. The only real plus is that VxVM is very easy to use when configured correctly and ports well into other Veritas products (VCS and Veritas File System, VxFS). A license without cluster support means that only one the servers can have the VxVM disk groups imported for use. This turned out to not be much of a problem due to the compatibility of VCS and VxVM. I'll address that later. The two servers are running Solaris 8 with the ufs file system. The SAN itself has sammfs as its file system. The clients who ultimately retrieve the data files are using ufs as their file systems. It is this progression of the data file going from ufs->sammfs->ufs that the size of the file must be kept as part of the metadata. My tasks were to install and configure VxVM version 3.5 to mirror the metadata disks, integrate the disk groups created by VxVM into an existing VCS configuration and secure VxVM by installing only the required packages and to ensure that the functionality was restricted in use to those users absolutely necessary. Additionally, I had to do all of this in the form of Perl scripts so that the process would be repeatable and every step documented, as those scripts were to end up in a configuration management system. My initial knowledge of VxVM was simply that I knew it was a product of Veritas and nothing more. I had previously attended a training class on VCS about a year prior to this task, but hadn't been able to apply the knowledge gained there to any system. Therefore, I knew very little of the existing VCS configuration on the target system. I'd spent that year working on firewalls. I had no idea where to start when I was ready to integrate the VxVM disk groups into the existing VCS configuration. Let me add that the VCS configuration was full of undocumented custom agents that I knew nothing about other than their existence. The only thing I really had on my side was a fair understanding of Perl and knowledge of the security requirements of my

program. One other thing…I was given a month to get this done.

During Snapshot

My preferred method of using a new product is to read as little as possible and go straight to the hands on approach. I've had great success with this method and have had several kernel panics, complete failures of the operating system, which resulted in reinstalling the operating system, and many other similarly devastating results. I do recommend reserving that method for your own personal equipment and not your employer's. Since this is my employer's equipment, I decided that the best place to start would be the Veritas Volume Manger™ 3.5 Administrator's Guide. Here, I learned that VxVM does so much more than what I was to ask of it. The information provided was pretty raw, unfriendly and certainly not written for this newbie. This posed a problem in the sense that I was in something of a time crunch and needed to get directly to the information concerning mirroring. I managed fairly well with the administrator's guide, but still had many questions that the guide simply didn't address. My questions loomed mostly around an apparent division in the way to accomplish volume configuration; vxassist or vxmake. The Internet was where I next went for those answers. There, as you might imagine, I was able to find a plethora of VxVM articles and tutorials. One of the most useful was from the Cuddletech website (4). Although they discuss a previous version of the product, I was able to solidify my understanding of the VxVM logical objects and gain a much better understanding of the vxmake and vxassist methods of configuring a system. The major difference between the two is that vxmake allows you complete control over object creation, but you must specify everything (and I mean everything) concerning that object in precise detail. The vxassist way makes a lot of decisions for you, but is much easier to use.

Installation and Patching VxVM

While researching and learning all I could, I had to keep in mind that I was to eventually script all of this and I had to find areas where I could harden this product as well as how to test the product to verify proper operation. The installation guide (5) provides instructions to install all of the VxVM packages and to in install them in a particular order. After examining each package and learning the functionality they provided, I wondered if only installing what I needed and omitting the others would be sufficient. I decided that all I needed was the licensing utility, the man pages and the actual VxVM binaries. Those packages are respectively VRTSvlic, VRTSvmman and VRTSvxvm. I have no need of adding the VxVM GUI packages, as the servers do not have video cards, which doesn't allow for a GUI. I do recommend using the GUI if it is at all possible. GUI management of the VxVM resources is significantly easier than

using the command line. VxVM management via the command line requires significant knowledge of several commands of which I'll discuss shortly. I manually added each package and then wrote a Perl script to do this for me. After extensive testing of the system, the three packages I chose to install worked as expected without any complications. So, my first step at product hardening was to remove any unneeded packages or in this case to never initially install them.

There is a program that VxVM requires to be executed before anything can be done with VxVM. That program is vxinstall. Then the root disk group, rootdg, must be created. It is a mandatory/default disk group and in my opinion is something of a dummy disk group, as VxVM doesn't allow you to deport it. What follows is a step-by-step account of installing VxVM, executing the vxinstall program and creating the root disk group.

1. Install necessary packages.

   ```
   % pkgadd -d . VRTSvlic VRTSvmman VRTSvxvm
   ```

   That completes the installation of the licensing utility, the man pages and the actual VxVM binaries.

2. Run the vxinstall utility.
   ```
   % vxinstall
   ```

   This is an interactive process. I have described the questions and provided the answers for my situation.

   - Are you prepared to enter a license key [y,n,q,?] (default: y) **y**
   - Enter your license key: **ABCD-EFGH-IJKL-MNOP-QRST-UVWX**
   - Do you wish to enter another license key [y,n,q,?] (default: n) **n**
   - Do you want to use enclosure based names for all disks [y,n,q,?] (default: n) **n**
   - Hit RETURN to continue. **<RETURN>**
   - Hit RETURN to continue. **<RETURN>**
   - Select an operation to perform: **3** (This prevents multipathing and suppresses devices from VxVM's view. This is desired in my case because I'm using SANtricity for load balancing and multipathing.)
   - Select and operation to perform: **5** (This prevents multipathing for all disks on a controller by VxVM.)
   - Enter a controller name [<ctlr-name>,all,list,list-exclude,q,?] **all**
   - Continue operation? [y,n,q,?] **y**
   - Hit RETURN to continue. **<RETURN>**
   - Select and operation to perform: **q** (This quits the vxinstall program.)

That completes the vxinstall program. You will notice in the example configuration file shown with the source code provided that where **<RETURN>** is specified above, an empty line exists in the configuration file.

3. Disable VxVM configuration daemon so that further manual configuration may be applied.

```
% vxconfigd -k -m disable
```

4. Initialize the volume configuration daemon.

```
% vxdctl init
```

5. Create the rootdg disk group.

```
% vxdg init rootdg
```

6. Allow VxVM to see device.

```
% vxdctl add disk c10t210d0
```

7. Write a header to the disk making useable by VxVM.

```
% vxdisksetup -i c10t210d0
```

8. Initialize the disk. This erases all information.

```
% vxdisk -f init c10t210d0
```

9. Add device to the rootdg disk group.

```
% vxdg adddisk c10t210d0
```

10. Enable the VxVM configuration daemon.

```
% vxdctl enable
```

11. Remove the file, install-db.

```
% rm /etc/vx/reconfig.d/state.d/install-db
```

The removal of this file will allow VxVM to come up after the reboot.

Note: For more information of the commands given here, there exists a man page on each of them on the system you are using after you have installed the VRTSvmman package.

Next, I needed to patch the newly installed binaries and script that as well. No problems here to mention. This is just standard Solaris patching with the patchadd command. The only item of note is that due to installing only what is absolutely necessary you will see entries in the patch log indicating failures. Verify that those failed entries are the result of that actual binary not being present on the system.

Configuring VxVM

As stated above, I chose the vxassist method to create my disk groups and volumes. After a trial and error phase with vxmake and vxassist, I decided I did not need the level of control over object creation offered by the vxmake utility. The scripts I've included with this document are configuration file dependent. Since the vxassist utility does so much behind the scenes, I was able to keep the configuration file contents to a minimum and reduce the complexity of those files. I have a personal goal when developing any code.

Below is an explanation of each step of the configuration process. The Perl scripts provided reflect the procedures given here.

1. Initialize the disks or in this case, what appears to the OS as disks.

   ```
   % vxdisksetup –i c10t210d5
   % vxdisksetup –i c12t240d3
   ```

   The targets listed above have been initialized and can now be seen by VxVM as useable disks. The targets shown here are actually being offered up by SANtricity, which is the software that manages the RAID volumes on the SAN.

   Note: There are two ways to incorporate disks into VxVM. The way I have chosen is to initialize the disks, which erases any data previously on them. The other method is to encapsulate the disks, which preserves any data previously residing on the disks.

2. Create the disk group, create VM disks and associate those objects with the newly created disk group.

   ```
   % vxdg init metaDG metadisk1=c10t210d5 metadisk2=c12t240d3
   ```

   The name of the disk group created is metaDG. The names of the VM disks are metadisk-01 and metadisk-02 and are associated to corresponding real disks.

3. Create the volume (vxassist).

```
% vxassist -g metaDG make metaDGvol 6291456 alloc=metadisk1
```

The volume size is 6291456, which is the number of 512 byte blocks. This allows for a total size of 3 GB for the volume.

What did vxassist just do? It created a sub-disk that it named metadisk1-01, associated that sub-disk to metadisk1, and created a plex it named metaDGvol-01. Then it created a volume that I named metaDGvol and placed the plex it created within that volume. Notice where vxassist names the objects it creates. When vxassist creates objects, the names it gives those objects always resemble the object they are most closely associated with.

4. Create the mirror (vxassist).

```
% vxassist -b -g metaDG mirror metaDGvol alloc=metadisk2 \
  init=none
```

The –b switch backgrounds the mirroring process, which takes about 10 minutes with this size volume on my system.

What did vxassist just do? It created a sub-disk it named metadisk2-01, associated that sub-disk to metatdisk2, and created a plex named metaDGvol-02. Then, it placed that plex into the metaDGvol volume, which is what makes it a mirror of the other plex. Notice how I didn't have to specify the size of the mirror. It takes the size of the volume it created in step 3 and uses that number. A note of clarity here concerning space; the disk group, metaDG, now consists of a total of 6 GB of space. The volume itself is just a logical object and will show a size of 3 GB as do each of the plexes it contains.

5. Wait for mirroring to complete.

```
% vxtask list
```

The command given above will display the percentage complete with respect to all existing VxVM tasks. It is extremely important that you allow the mirroring to complete before doing anything with the disk group. Attempting to access this disk group before the mirroring completes will cause the mirroring to fail. This bit of information came to me painfully and took while to figure out. In the process of my trial and error scripting, I had created the disk group resource within VCS, but had the resource offline within the VCS configuration. I knew that VCS verified periodically that online resources were still online, but wasn't aware that VCS verifies that resources are offline as well. My mirroring kept failing and I was convinced it was my suspect scripting skills. It turned out that when VCS went to verify the disk group was offline and found it online it issued a

'vxdg deport metaDG' command. Not good. That command does what it sounds like and removes the resource from the operating systems view, which stops all activity to this disk group. The fix is to disable the resource within the VCS configuration.

6. Set up DRL (vxassist).

```
% vxassist -g metaDG addlog metaDGvol logtype=drl nlog=2 \
      alloc=metadisk1,metadisk2
```

What did vxassist just do? It created sub-disks on each of the specified VM disks, allocated the appropriate space for the volume size on each sub-disk (this is the DRL) and placed the sub-disks within the metaDGvol volume.

That's it. The configuration of the disk group and its resources is complete. The targets are mirrored and Dirty Region Logging is setup. Next, I'll explain how I incorporated the VxVM resources into VCS. In the event that manual interaction with these resources is required, importing, deporting the disk group and starting/stopping the volume can be accomplished as such:

First, import the disk group.

```
% vxdg import metaDG
```

This brings the resource into the operating systems view. Next the volume needs to be started.

```
% vxvol start metaDGvol
```

To make use of the volume, it must be mounted.

```
% mount /dev/vx/dsk/metaDG/metaDGvol /metamirr
```

The above command mounts the resource to the directory metamirr. That directory can now be written to and accessed as any other directory.

To remove this resource, simply reverse the above procedures.

```
% unmount /metamirr
% vxvol stop metaDGvol
% vxdg deport metaDG
```

VCS/VxVM Inc.

With the VxVM configuration complete, the next task was to incorporate these VxVM resources in to an existing VCS configuration. There is ample information on setting up VxVM, but not incorporating it into VCS. I had another problem with these new VxVM resources in that the license I was given didn't support cluster functionality, which meant I couldn't share the metaDG disk group out. Only one server at a time could have access to this disk group, but I needed either server to be able to have this access (just not at the same time). This is where VCS really shined. Since both products are from Veritas, I suspected that they provided built in support for their other products within VCS. They do,

in fact, have a disk group resource that provided the functionality I needed. All I had to do was to create the disk group resource and supply metaDG for the disk group *name* property. The resource agent, which manages the resource, knows how to import and deport the disk group. Only one of my two servers needed to see this disk group at a time. So, if VCS were to failover the disk group from one server to another, it would deport the disk group on one server and import on the other. Make sure that you deport the disk group using the procedures shown above before having VCS take control. The disk group resource agent is expecting the disk group deported when it takes control.

VxVM Security

As stated above, I had two security requirements concerning VxVM.

1. Install only what was absolutely necessary. I was able to do that by installing only the necessary packages.

2. Restrict usage of product to the minimum number of users. I needed to ensure that what VxVM functionality was left to utilize was limited to the minimum number of users. The only user that I believed required access was the root user. So, I modified the permissions so that only root could use any of the VxVM binaries.

As with any system, this system isn't unbreakable and there may well be more I could do and will if that information comes my way, but by not installing the remaining VxVM packages I haven't opened the system up to the existing or future vulnerabilities of those packages. Unless someone gains root access, the common user can't utilize the VxVM functionality. A common user wouldn't even be able to issue a vxprint command to see what resources VxVM provides.

After Snapshot

Conclusion

Ultimately, the task was completed. The benefits gained from adding VxVM are zero down time in the event a single device is lost from the operating systems view and the ability of VCS to control the resources provides an additional assurance of information availability. The security requirements were satisfied and although no product can really be considered unbreakable it is fairly well hardened and the risk of compromise has been reduced. The availability of the metadata storage has been increased by a factor of at least two by having the data mirrored in two places and by VxVM not interrupting the I/O in the event

one of the two plexes fail. Creating DRL logs will reduce volume recovery time in the event of failure.

Source Code

The scripts provided will install, patch and configure VxVM and should be portable to any UNIX based host requiring only edits to the configuration files. There are a total of three scripts. A reboot of the system is required after completion of the first two scripts. The reboots are required as the VxVM binaries actually plug into the OS kernel and that only happens when the system is rebooted. The scripts are configuration file dependent. This should allow an administrator the ability to simply modify a configuration file and leave the source code alone. Creating the scripts with configuration file dependency proved extremely valuable in testing as some of the volumes would succeed during the mirroring process and others would fail. I was able to just comment out the successful entries and retry the failed entries.

REFERENCES

1. Veritas Volume Manager™ 3.5 Administrator's Guide – Solaris. August 2002. <http://www.sun.com/products-n-solutions/hardware/docs/pdf/875-3356-10.pdf>
2. "Storage Area Networking". 2003. Unixway, LLC. <http://www.unixway.com/san/>
3. "Dictionary.com/metadata". 2005. Lexico Publishing Group, LLC. <http://dictionary.reference.com/search?q=metadata>
4. "VXVM Kickstart – Enterprise Storage Management, Cuddletech Style". Rockwood, Ben. <http://www.cuddletech.com/veritas/>
5. Veritas Volume Manager™ 3.5 Installation Guide. July 2002. <http://www.sun.com/products-n-solutions/hardware/docs/pdf/875-3355-10.pdf>
6. Veritas Krash Kourse: The Land of Who's Who of Vx Land. August 2002. Rockwood, Ben. <http://www.cuddletech.com/veritas/vxcrashkourse/vxcrashkourse.pdf>

Source code for install VxVM:

```
#!/usr/bin/perl –w

# filename: install_VxVM.pl

use strict;
$| = 1;

print "Beginning install_VxVM.pl\n";

########################################
## Add VxVM packages ###################
########################################

print "Adding package: VRTSvlic\n";
systemWrapper("pkgadd –a ./noask_pkgadd –d . VRTSvlic");

print "Adding package: VRTSvxvm\n";
open FH, "| pkgadd –a ./noask_pkgadd –d . VRTSvxvm"
       or die "ERROR: Can't open pipe into pkgadd for VRTSvxvm: $!\n";
print FH "8\n";
print FH "y\n";
close FH;

print "Adding package: VRTSvmman\n";
systemWrapper("pkgadd –a ./noask_pkgadd –d . VRTSvmman");

# run vxinstall
print "Beginning vxinstall\n";
systemWrapper("usr/sbin/vxinstall < response.txt");
print "Completed vxinstall\n";

# create the root diskgroup
my $rootdisk = `cat ./rootdisk.txt`;  chomp $rootdg;
systemWrapper("/usr/sbin/vxconfigd –k –m disable");
systemWrapper("/usr/sbin/vxdctl init");
systemWrapper("/usr/sbin/vxdg init rootdg");
systemWrapper("/usr/sbin/vxdctl add disk $rootdisk");
systemWrapper("/etc/vx/bin/vxdisksetup –I $rootdisk");
systemWrapper("/usr/sbin/vxdisk –f init $rootdisk");
systemWrapper("/usr/sbin/vxdg adddisk $rootdisk");
systemWrapper("/usr/sbin/vxdctl enable");
systemWrapper("rm /etc/vx/reconfig.d/state.d/install-db");

print "Completed install_VxVM.pl\n";

exit 0;

##################
## Subroutines ##
##################

sub systemWrapper {
  my $cmd = shift @_;
  my $ret = system "$cmd";
  if($ret) {
    print "ERROR: Problem with $cmd: $!\n";
    exit 1;
  } else { print "completed: $cmd\n"; }
}
#### END of install_VxVM.pl ###################################
Configuration files used by install_VxVM.pl
```

filename: response.txt

```
y
ABCD-EFGH-IJKL-MNOP-QRST-UVWX
n
n


3
y
5
all
y

q
```

filename: rootdisk.txt – This is an example. You should insert whatever device you have available for the root disk group in place of what is given below.

```
c10t210d0
```

Source code for patching VxVM:

```
#!/usr/bin/perl -w

# filename: patch_VxVM.pl
# This script will apply the patches listed in VxVM_patches.txt

use strict;
$| = 1;

print "Beginning patch_VxVM.pl\n";

my $ret;
my $patch;
open FH, "VxVM_patches.txt" or die "ERROR opening VxVM_patches.txt for reading: $!\n";
foreach (<FH>) {
  chomp;
  print "Adding patch: $_\n";
  $ret = system "patchadd $_";
  if ($ret) {
    print "ERROR with adding patch # $_: $!\n";
    exit 1;
  }
}

print "Completed patch_VxVM.pl\n";

exit 0;

#### END  ########################################
```

Configuration files used by patch_VxVM.pl

filename: VxVM_patches.txt

```
112392-06
```

Source code for configuring VxVM:

```
#!/usr/bin/perl -w

# filename: config_VxVM.pl

use strict;
$| = 1;

my $vxassist    = "/usr/sbin/vxassist";
my $vxdisksetup = "/etc/vx/bin/disksetup";
my $vxdctl      = "/usr/sbin/vxdctl";
my $vxdg        = "/usr/sbin/vxdg";

print "Beginning config_VxVM.pl\n";

# See if invoked with 'log' option; setup logging if so.
if ($ARGV[0] eq "log") {
  print "Setting up logging.\n";
  systemWrapper("cp /etc/init.d/vxvm-sysboot /etc/vxvm-sysboot.ORIG");
  open FH, "etc/init.d/vxvm-sysboot"
              or die "ERROR opening vxvm-sysboot for reading: $!\n";
  my @log_lines = <FH>;  chomp @log_lines;
  open FH, ">/etc/init.d/vxvm-sysboot"
              or die "ERROR opening vxvm-sysboot for writing: $!\n";
  foreach (@log_lines) {
    s/^#opts="\$opts -x syslog/opts="\$opts -x syslog/;
    s/^#debug=1/debug=9/;
    print FH "$_\n";
  }

# read in configuration file
open FH, "mirrors.conf" or die "ERROR opening mirrors.conf for reading: $!\n";
my @conf_lines;
foreach (<FH>) { push @conf_lines, $_ if (!/^#|^\s*$/); }

my $dsk_grp, $vol, $vol_sz, $vm_dsk1, $vm_dsk2, $dsk1, $dsk2;

# Need to allow VxVM to see the newly added disks from the SAN.
systemWrapper("$vxdctl enable");

# initialize disks, create volumes, then create mirrors (vxassist way)
foreach (@conf_lines) {
  ($dsk_grp,$vol,$vol_sz,$vm_dsk1,$dsk1,$vm_dsk2,$dsk2) = split;
  for (1..2) { systemWrapper("/etc/vx/bin/vxdisksetup -f -i $dsk$_"); }
  systemWrapper("$vxdg init $dsk_grp ${vm_dsk1}=$dsk1 ${vm_dsk2}=$dsk2");
  systemWrapper("$vxassist -g $dsk_grp make $vol $vol_sz alloc=$vm_dsk1");
  systemWrapper("$vxassist -b -g $dsk_grp mirror $vol alloc=$vm_dsk2 init=none");
}

# wait for mirroring to complete  ( a cheap timer )
while(1){
  sleep(10);  # sleeping first gives VxVM time to get the mirroring process started
  my @tmp = `vxtask list`;
  last if ((@tmp == 1) && (print "\nMirroring complete.\n";));
  print "\nMirroring status…\n";
  print @tmp;
}

# set up dirty region logging
foreach (@conf_lines) {
  ($dsk_grp,$vol,$vol_sz,$vm_dsk1,$dsk1,$vm_dsk2,$dsk2) = split;
  systemWrapper("$vxassist -g $dsk_grp addlog $vol logtype=drl nlog=2
                           alloc=$vm_dsk1,$vm_dsk2");

print "\nCompleted config_VxVM.pl\n";
```


© SANS Institute 2005 Author retains full rights.

```
exit 0;

##############################
## SUBROUTINES ###############
##############################

sub systemWrapper {
  my $cmd = shift @_;
  my $ret = system "$cmd";
  if ($ret) {
     print "ERROR: problem with: $cmd\n";
     exit 1;
  } else {
     print "complete: $cmd\n";
  }
}


#### END  ###########################################################
```

Configuration file format for config_VxVM.pl

Filename: mirrors.conf

```
#dg_name vol_name size vmdisk1 vmdisk2 disk1 disk2
metaDG metaDGvol 6291456 metadisk1 metadisk2 c10t210d5 c12t240d3
```