How to Setup and Secure NFS

Motivation

Depending on what you are trying to accomplish, sometimes the old methods are still the best. At work recently, we discovered that we needed to share data between several nodes. We could have used a web based service to do this, but it would have taken more time to write this than implementing Network File Share (NFS) and sharing out the local data to all the nodes. So this paper was written as a resource so I don't have to figure out how to implement this solution again. Incidentally, NFS is really simple, so if you forget, it's not hard to get all of the pieces right. One of the must haves with NFS is security, since it relies on IP addresses as its only method of protecting access to its services.

History

'Network File System is a distributed file system protocol originally developed by Sun Microsystems in 1984, allowing a user on a client computer to access files over a network much like local storage is accessed. NFS, like many other protocols, builds on the Open Network Computing Remote Procedure Call (ONC RPC) system. The Network File System is an open standard defined in RFCs, allowing anyone to implement the protocol.

Versions and variations

Sun used version 1 only for in-house experimental purposes. When the development team added substantial changes to NFS version 1 and released it outside of Sun, they decided to release the new version as v2, so that version interoperation and RPC version fallback could be tested.

NFSv2

Version 2 of the protocol (NFSv2) is defined in RFC 1094, released in March 1989. People involved in the creation of NFS version 2 include Russel Sandberg, Bob Lyon, Bill Joy, Steve Kleiman, and others.

NFSv2 originally operated only over UDP. Its designers meant to keep the server side stateless, with locking (for example) implemented outside of the core protocol. The decision to make the file system stateless was a key decision, since it made recovery from server failures trivial (all network clients would freeze up when a server crashed, but once the server repaired the file system and restarted, all the state to retry each transaction was contained in each RPC, which was retried by the client stub(s).) This design decision allowed UNIX applications (which could not tolerate file server crashes) to ignore the problem.

The Virtual File System interface allowed a modular implementation, reflected in a simple protocol. By February 1986, implementations were demonstrated for operating systems such as System V release 2, Microsoft DOS, and VAX/VMS using Eunice.

Due to 32-bit limitations, NFSv2 allowed only the first 2 GB of a file to be read.

NFSv3

Version 3 (RFC 1813, June 1995) added:

- support for 64-bit file sizes and offsets, to handle files larger than 2 gigabytes (GB);
- support for asynchronous writes on the server, to improve write performance;
- additional file attributes in many replies, to avoid the need to re-fetch them;

- a READDIRPLUS operation, to get file handles and attributes along with file names when scanning a directory;
- assorted other improvements.

At the time of introduction of Version 3, vendor support for TCP as a transport-layer protocol began increasing. While several vendors had already added support for NFS Version 2 with TCP as a transport, Sun Microsystems added support for TCP as a transport for NFS at the same time it added support for Version 3. Using TCP as a transport made using NFS over a WAN more feasible.

NFSv4

Version 4 (RFC 3010, December 2000; revised in RFC 3530, April 2003), influenced by AFS and CIFS, includes performance improvements, mandates strong security, and introduces a stateful protocol. Version 4 became the first version developed with the Internet Engineering Task Force (IETF) after Sun Microsystems handed over the development of the NFS protocols.

NFS version 4.1 (RFC 5661, January 2010) aims to provide protocol support to take advantage of clustered server deployments including the ability to provide scalable parallel access to files distributed among multiple servers (pNFS extension). NFS version 4.2 is currently being developed.

Not everyone was happy with the new protocol. In 2010, OpenBSD's Theo de Raadt wrote: "NFSv4 is not on our roadmap. It is a ridiculous bloated protocol which they keep adding crap to."

Platforms

'NFS is often used with Unix operating systems (such as Solaris, AIX and HP-UX) and Unix-like operating systems (such as Linux and FreeBSD). It is also available to operating systems such as the classic Mac OS, OpenVMS, IBM i, certain editions of Microsoft Windows, and Novell NetWare. Alternative remote file

access protocols include the Server Message Block (SMB, also known as CIFS), Apple Filing Protocol (AFP), NetWare Core Protocol (NCP), and OS/400 File Server file system (QFileSvr.400).

SMB and NetWare Core Protocol (NCP) occur more commonly than NFS on systems running Microsoft Windows, AFP occurs more commonly than NFS in Macintosh systems, and QFileSvr.400 was once found more commonly in IBM i systems. Haiku recently added NFSv4 support as part of a Google Summer of Code project.'

Source: http://en.wikipedia.org/wiki/Network_File_System

Competitors

'In computer networking, Server Message Block, one version of which was also known as Common Internet File System (CIFS, /ˈsɪfs/), operates as an application-layer network protocol mainly used for providing shared access to files, printers, serial ports, and miscellaneous communications between nodes on a network. It also provides an authenticated inter-process communication mechanism. Most usage of SMB involves computers running Microsoft Windows, where it was known as "Microsoft Windows Network" before the subsequent introduction of Active Directory. Corresponding Windows services are LAN Manager Server (for the server component) and LAN Manager Workstation (for the client component).

SMB can run on top of the Session (and lower) network layers in several ways:

Directly over TCP, port 445;

Via the NetBIOS API, which in turn can run on several transports:

On UDP ports 137, 138 & TCP ports 137, 139 ([NetBIOS over TCP/IP]]);

On several legacy protocols such as NBF (incorrectly referred to as NetBEUI).

The SMB "Inter-Process Communication" (IPC) system provides named pipes and was one of the first inter-process mechanisms commonly available to programmers that provides a means for services to inherit the authentication carried out when a client first connected to an SMB server.

Some services that operate over named pipes, such as those which use Microsoft's own implementation of DCE/RPC over SMB, known as MSRPC over SMB, also allow MSRPC client programs to perform authentication, which over-rides the authorization provided by the SMB server, but only in the context of the MSRPC client program that successfully makes the additional authentication.

SMB signing: Windows NT 4.0 Service Pack 3 and upwards have the capability to use cryptography to digitally sign SMB connections. The most common official term is "SMB signing". Other terms that have been used officially are "[SMB] Security Signatures", "SMB sequence numbers" and "SMB Message Signing". SMB signing may be configured individually for incoming SMB connections (handled by the "LanManServer" service) and outgoing SMB connections (handled by the "LanManWorkstation" service). The default setting from Windows 98 and upwards is to opportunistically sign outgoing connections whenever the server also supports this. And to fall back to unsigned SMB if both partners allow this. The default setting for Windows domain controllers from Server 2003 and upwards is to not allow fall back for incoming connections. The feature can also be turned on for any server running Windows NT 4.0 Service Pack 3 or later. This protects from man-in-the-middle attacks against the Clients retrieving their policies from domain controllers at login.

The design of Server Message Block version 2 (SMB2) aims[citation needed] to mitigate this performance-limitation by coalescing SMB signals into single packets.

SMB supports opportunistic locking — a special type of locking-mechanism — on files in order to improve performance.

SMB serves as the basis for Microsoft's Distributed File System implementation.'

Source: http://en.wikipedia.org/wiki/Network_File_System

Future work

- An enormous update (more than 600 pages), NFSv4.1 became a standard in 2010, and included optional pNFS (parallel) support and better cifs/windows ACL interoperability
- But NFS v4.1 Implementations have lagged, and even NFSv4 deployment slower than expected. Most Linux NFS users still use 17 year old NFS version 3, even though it is not fully posix compliant and can not do safe caching
- Work continues on an opensource userpsace Ganesha NFS server with NFSv4.1/pNFS support, and also the kernel pNFS kernel server (currently maintained out of kernel). Few pNFS clients other than Linux exist.

Source: http://2014.texaslinuxfest.org/sites/default/files/smf-texas-linux-fest-network-fs-futures_0.pdf

Setup and deployment

In the following examples, the Server is using IP address 192.168.139.50. The clients range from 192.168.139.51-55. All computers in this example are running CentOS 6.5. Because of the font, most of the lines below wrap around to the next line. You should be able to cut and paste into a putty session to get these right, then modify for your local IP addresses. In Microsoft Windows, Notepad is an excellent way to remove special formatting. It might benefit you to cut and paste into Notepad, then move from there to your SSH session (e.g. PuTTy).

Server Side (Computer sharing data):

Create a directory to share data from:

```
mkdir -p /staging/nfs
```

Install the necessary binaries and their libraries:

```
yum install nfs-utils.x86_64 nfs4-acl-tools.x86_64 nfs-utils-lib-devel.x86_64
nfs-utils-lib.x86_64 -y
```

Modify /etc/fstab and append:

```
/dev/sdb1          /staging/nfs              ext4  defaults,noatime 1 2
```

Modify /etc/exports and add:

```
/staging/nfs       192.168.139.0/24(rw,async,no_root_squash,no_subtree_check)
```

Modify /etc/hosts.allow and append:

```
# this is an example:
lockd:   192.168.139.51 , 192.168.139.52 , 192.168.139.53 , 192.168.139.54 ,
192.168.139.55
mountd:  192.168.139.51 , 192.168.139.52 , 192.168.139.53 , 192.168.139.54 ,
192.168.139.55
portmap: 192.168.139.51 , 192.168.139.52 , 192.168.139.53 , 192.168.139.54 ,
192.168.139.55
rquotad: 192.168.139.51 , 192.168.139.52 , 192.168.139.53 , 192.168.139.54 ,
192.168.139.55
statd:   192.168.139.51 , 192.168.139.52 , 192.168.139.53 , 192.168.139.54 ,
192.168.139.55
```

Modify /etc/hosts.deny and append:

```
#
portmap:ALL
lockd:ALL
mountd:ALL
rquotad:ALL
statd:ALL
```

Activate the service and setup for reboot to turn on:

```
for XX in netfs nfs nfslock portreserve rpcbind rpcgssd rpcidmapd rpcsvcgssd;
do chkconfig --level 35 ${XX} on; service ${XX} restart; done
```

Testing:

```
showmount -e localhost
```

Client Side (Computer accessing data):

Install the necessary binaries and their libraries:

```
yum install nfs-utils.x86_64 nfs4-acl-tools.x86_64 nfs-utils-lib-devel.x86_64
nfs-utils-lib.x86_64 -y
```

Create the directory that will mount to the NFS server:

```
mkdir -p /staging/nfs
```

Edit /etc/fstab and add:

```
192.168.139.60:/staging/nfs  /staging/nfs   nfs
auto,noatime,nolock,bg,nfsvers=3,intr,tcp,actimeo=1800 0 0
```

Mount the remote file system:

```
mount -a
```

Check that the file system mounted:

```
df -h
```

Conclusion

In the preceding examples, I have laid out how to setup and secure NFS. The security is in two places, one in /etc/exports where we define the subnet that is allowed to access that server (i.e. 192.168.139.0/24). And two, in the combination of /etc/hosts.allow and /etc/hosts.deny. Since NFS has TCP Wrapper encoded into itself (meaning that it can utilize TCP Wrapper), this enhancement allows the administrator to be very concise on exactly what IP addresses are allowed to connect to this server. In the above example for TCP Wrapper, I have allowed all other protocols into the server, only restricting those that are relevant to NFS on the NFS Server. Further work could be done with utilizing iptables to secure the protocol. I have mixed feelings with iptables. I love what it does and how it works, but for an enterprise server farm, I am reluctant to implement it on the inside servers because of the work load required to maintain the rules. That in my opinion is what the external and internal firewalls are supposed to do, offload the ACLs (rules) from the routers. By implementing iptables, you are incurring a tax for the overhead I/O that is used against every packet coming into the server. However, I have no restraints from using iptables when deploying web services in the DMZ where my trust level of my servers is not 100% definite of the risk levels levied against said.

Resources

History of NFS:
http://en.wikipedia.org/wiki/Network_File_System

Future of NFS:
http://2014.texaslinuxfest.org/sites/default/files/smf-texas-linux-fest-network-fs-futures_0.pdf

History of TCP wrapper:
http://en.wikipedia.org/wiki/TCP_Wrapper