The Network File System

- One of the biggest challenges in distributed, heterogeneous networks is giving users access to the files and applications they require over a network.

- **Network File Sharing** (**NFS**) together with **Samba** allows UNIX and Windows users to easily access files on either platform.

- NFS hides underlying details about where files are physically stored. It allows files that are physically located elsewhere to appear as if they are part of a local file system.

- NFS is both a standard specification and a set of software products that enable file access across a network.

- It primarily utilises two different protocol mechanisms: the **External Data Representation** (**XDR**) protocol, which operates at the presentation layer, and **Remote Procedure Calls** (**RPC**), which operate at the session layer.

- These two protocols are the foundation for all NFS interaction. RPC provides the basis for message exchanges between clients and servers.

- The XDR protocol provides data translation between different types of computers and operating systems.

- From a user perspective, NFS is transparent. Depending on how NFS has been set up, users may be able to log onto any workstation in their network and see their files. Files that are physically under a different operating system are easily accessed as is they existed locally.

- From a system administrator standpoint, NFS is a distributed file system. An NFS server has one or more file systems that are mounted by NFS clients.

- To NFS clients, remote disks look like local disks. This also means that files on an NFS server can be accessed by many different NFS client computers, thus centralizing file distribution.

Technical Overview

- NFS is both scalable and flexible. It supports small to large networks, as well as a wide range of hardware and operating systems.

- It can effectively support small networks of less than 10 clients or large networks of more than 25,000 clients, physically scattered over the world.

- An NFS server is a host that owns one or more file systems and makes them available on a network.

- NFS clients then mount file systems from one or more servers. In addition, NFS provides file locking services, which control multiple accesses to the same file.

- One of the key functions NFS software performs when transferring files between Windows and UNIX is preserving file permissions.

Windows Permissions

- Under Windows NTFS, each file and directory is owned by one account. The owner of this account is the only one who has the right to access, modify, and secure it from outside access. By default, the owner is the user who created the resource.

- An exception to this is a user who is in the Administrator group. In this case, all users who are in the Administrator group co-own any resource created by any group member.

- In addition to permissions, each file and directory has at least one access control list (ACL) entry that defines a user's or group's access to a file or directory. By using multiple ACLS, you have a great deal of control over who has access to what.

- The owner of a file or directory may grant a user the following types of access:

Type of Access	File Type	Description
No Access	directory or file	User cannot access the file or directory by any means.
List	directory only	User can view the contents of a directory, but cannot access those contents.
Read	directory or file	User can read files or execute programs, but cannot modify them in any way.
Add	directory only	User can write files to a directory, but cannot view or read the contents of the directory.
Add and Read	directory or file	User can view, read, and save new files, but cannot modify existing files.
Change	directory or file	User can view, read, execute, or save new files; modify or delete existing files; change file attributes; or delete the directory.
Full Control	directory or file	User can read, execute, modify, delete, change permissions, or take ownership away from the current owner.
Special Access	directory or file	Allows the owner or any user granted "P" permission to custom-build an access control entry for an ACL.

UNIX Permissions

- In UNIX there are three categories of access permissions: **user**, **group**, and other (also known as **world**) for files and directories.
- Each category is set to individually enable or disable **read** (**r**), **write** (**w**), and **execute** (**x**) privileges.
- UNIX systems also store a numeric owner (UID) and group (GID) identifier for each file or directory. Together, access permissions and identifiers determine a user's access to a file or directory.
- UNIX permissions also determine the file type. For example, even it a file is really an executable it will not run unless the execute (x) flag has been set.
- The owner of a file or directory may grant the following types of access to themselves (owner), their group (group), or everyone else (other or world):

Type of Access	**File Type**	**Description**
Read	directory or file	User can read a file or view contents of a directory.
Write	directory or file	User can modify or delete a file or directory. They can also create a new file or directory.
Execute	directory or file	User can ***cd*** to a directory, or execute a file.

- Owners typically have **rwx** access, while group and world access is limited. However, the file owner can change this to be anything they want.

File Mapping

- When you transfer files between UNIX and NT using NFS, file permissions must be mapped to equivalents on each system. The first step is to authenticate the user's identity.
- NFS software attempts to translate the user login ID to an equivalent login ID on the other system. If NFS can't translate the login ID, the next step is to use a translation table. If there isn't a translation, the NFS software can either assign an ID or end the connection.
- When a UNIX user creates a file under NTFS, the NFS software often uses NTFS's special access permission flags rather than standard NTFS access types (Add, Full Control, etc.).
- This allows NFS to assign the appropriate access to files. For example, under UNIX, write permission also implies delete. Depending on the NT NFS product you choose, the software may set Special Access flags for both write and delete.
- Since NFS is the de facto method of sharing file systems under UNIX, virtually all UNIX systems are already capable of functioning as either an NFS client or NFS server.

Exporting File Systems

- Virtually all versions of UNIX use a text configuration file for specifying the file systems that will be made available via NFS. In most cases, this file is named ***/etc/exports***.
- The ***/etc/exports*** file lists the file systems that a UNIX system makes available via NFS. In addition to listing the file systems, it also contains access permissions and restrictions for those file systems.
- NFS does not advertise its exported file systems the way some other operating systems do. Instead, it maintains a list of currently exported file systems, which it uses to respond to mount requests from NFS clients.
- Most UNIX file systems or subdirectories can be exported via NFS, with a couple of exceptions. Only local file systems can be exported; you cannot export a remote file system via NFS.
- Also, if you try to export a subdirectory or a parent directory of a file system that is already exported, the file system that you are trying to export must be on a different physical partition or device.
- Thus, if ***/usr/local*** is exported via NFS, you cannot export ***/usr/local/bin*** unless it exists on a different partition or device.
- Exporting file systems via NFS is actually a two-step process. First, you must create the appropriate entries in the ***/etc/exports*** file.
- Then you must tell UNIX to actually make the file systems available using the ***exportfs*** command.
- The NFS implementation under Red Hat Linux is similar to that of any Unix OS.
- File systems to be exported via NFS are listed in ***/etc/exports***.
- Shared directories are accessed through the ***mount*** command.

Client Side

- Must have the ***portmap*** daemon running.
- Performs the mount. For example:

```
mount -t nfs servername:/home/ftp/pub /mnt/directory
```

- **mount** associates a shared directory on the network with a mount point in your local filesystem.
- The above command will mount an exported directory from an NFS server and access it as if it were local to your machine.

Server Side

- The server must have three daemons running:

  portmap: maps calls made from other machines to the correct NFS daemon.
  rpc.nfsd: translates NFS requests into requests on the local filesystems.
  rpc.mountd: mounts and unmounts filesystems.

- To verify that these services are running, use:

  ```
  rpcinfo -p nfsserver
  ```

- Filesystems to be exported via NFS are defined in ***/etc/exports***. Here is an example:

  ```
  /home/ftp/pub              *.milliways.bcit.ca(ro) beetelgeuse.bcit.ca(rw)
  /home/nfstest/documents    zaphod.magrathea.com(rw)
  /data                              192.168.113.0/255.255.255.0
  ```

- Each entry specifies one exported directory and its access permissions.
- Hostnames can contain wildcards, as shown above.
- IP addresses of hosts can be specified individually or using a network/netmask specification.
- The option listing for each machine will describe what kind of access that machine will have:

  - **ro**: The directory is shared read only; the client machine will not be able to write to it. This is the default.
  - **rw**: The client machine will have read and write access to the directory.
  - **no_root_squash**: By default, any file request made by user root on the client machine is treated as if it is made by user nobody on the server. If **no_root_squash** is selected, then root on the client machine will have the same level of access to the files on the system as root on the server.
  - This can have serious security implications, although it may be necessary if you want to perform any administrative work on the client machine that involves the exported directories. Use this option with caution.
  - **no_subtree_check**: If only part of a volume is exported, a routine called subtree checking verifies that a file that is requested from the client is in the appropriate part of the volume. If the entire volume is exported, disabling this check will speed up transfers.
  - **sync**: By default the **exportfs** command will use **async** behavior, telling a client machine that a file write is complete - that is, has been written to stable storage - when NFS has finished handing the write over to the filesysytem. This behavior may cause data corruption if the server reboots, and the **sync** option prevents this.

- Remember that you must tell UNIX to actually make the file systems available using the ***exportfs*** command:

```
/usr/sbin/exportfs  -v
```

<u>Restricting Access</u>

- The **/etc/hosts.allow** and **/etc/hosts.deny** files specify which computers on the network can use services on your machine on a per-service basis.
- Each line of the file contains a single entry listing a service and a set of machines.
- When the server gets a request from a machine, it first checks **hosts.allow** to see if the machine matches a description listed in there. If it does, then the machine is allowed access.
- If the machine does not match an entry in **hosts.allow**, the server then checks **hosts.deny** to see if the client matches a listing in there. If it does then the machine is denied access.
- If the client matches no listings in either file, then it is allowed access.
- In addition to controlling access to services handled by ***xinetd***, this file can also control access to NFS by restricting connections to the daemons that provide NFS services.
- The first daemon to restrict access to is the **portmapper**. This daemon essentially just tells requesting clients how to find all the NFS services on the system.
- Restricting access to the portmapper is the best defense against someone breaking into your system through NFS because completely unauthorized clients won't know where to find the NFS daemons.
- A good approach to network security in general is to explicitly deny access to all services from all IP addresses in the **hosts.deny** file and then selectively grant access using the **hosts.allow** file. This is known as the paranoid approach.
- The first step in doing this is to add the followng entry to **/etc/hosts.deny**:

```
ALL:ALL
```

- A more permissive approach to security would be the following entries in the file (specific to NFS):

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

- The next step is to add an entry to **hosts.allow** to give grant service access to hosts that are trusted hosts. The format for hosts.allow file is:

```
service: host [or network/netmask] , host [or network/netmask]
```

- The following example illustrates how we can grant NFS access to two machines: 192.168.0.1 and 192.168.0.2, respectively:

```
portmap: 192.168.0.1 , 192.168.0.2
lockd: 192.168.0.1 , 192.168.0.2
rquotad: 192.168.0.1 , 192.168.0.2
mountd: 192.168.0.1 , 192.168.0.2
statd: 192.168.0.1 , 192.168.0.2
```

Starting and Verifying NFS services

- The service can be started using the standard script:

  /etc/rc.d/init.d/nfs start

- Verify that the service is running, query the server by executing the following command:

  rpcinfo -p

- You should get an output similar to:

```
program vers proto   port
100000    2   tcp    111  portmapper
100000    2   udp    111  portmapper
100011    1   udp    749  rquotad
100011    2   udp    749  rquotad
100005    1   udp    759  mountd
100005    1   tcp    761  mountd
100005    2   udp    764  mountd
100005    2   tcp    766  mountd
100005    3   udp    769  mountd
100005    3   tcp    771  mountd
100003    2   udp   2049  nfs
100003    3   udp   2049  nfs
300019    1   tcp    830  amd
300019    1   udp    831  amd
100024    1   udp    944  status
100024    1   tcp    946  status
100021    1   udp   1042  nlockmgr
100021    3   udp   1042  nlockmgr
100021    4   udp   1042  nlockmgr
100021    1   tcp   1629  nlockmgr
100021    3   tcp   1629  nlockmgr
100021    4   tcp   1629  nlockmgr
```