SAMBA Project Documentation

Table of Contents

Table of Contents

SAMBA Project Documentation

SAMBA Team

Table of Contents

Chapter 1. How to Install and Test SAMBA

Step 0: Read the man pages

The man pages distributed with SAMBA contain lots of useful info that will help to get you started. If you don't know how to read man pages then try something like:

```
$ nroff -man smbd.8 | more
```

Other sources of information are pointed to by the Samba web site, http://www.samba.org

---

Step 1: Building the Binaries

To do this, first run the program **./configure** in the source directory. This should automatically configure Samba for your operating system. If you have unusual needs then you may wish to run

```
root# ./configure --help
```

first to see what special options you can enable. Then exectuting

```
root# make
```

will create the binaries. Once it's successfully compiled you can use

```
root# make install
```

to install the binaries and manual pages. You can separately install the binaries and/or man pages using

```
root# make installbin
```

and

```
root# make installman
```

Note that if you are upgrading for a previous version of Samba you might like to know that the old versions of the binaries will be renamed with a ".old" extension. You can go back to the previous version with

```
root# make revert
```

if you find this version a disaster!

Step 2: The all important step

At this stage you must fetch yourself a coffee or other drink you find stimulating. Getting the rest of the install right can sometimes be tricky, so you will probably need it.

If you have installed samba before then you can skip this step.

Step 3: Create the smb configuration file.

There are sample configuration files in the examples subdirectory in the distribution. I suggest you read them carefully so you can see how the options go together in practice. See the man page for all the options.

The simplest useful configuration file would be something like this:

```
[global]
   workgroup = MYGROUP

   [homes]
      guest ok = no
      read only = no
```

which would allow connections by anyone with an account on the server, using either their login name or "homes" as the service name. (Note that I also set the workgroup that Samba is part of. See BROWSING.txt for defails)

Note that **make install** will not install a `smb.conf` file. You need to create it yourself.

Make sure you put the smb.conf file in the same place you specified in the`Makefile` (the default is to look for it in `/usr/local/samba/lib/`).

For more information about security settings for the [homes] share please refer to the document UNIX_SECURITY.txt.

Step 4: Test your config file with testparm

It's important that you test the validity of your `smb.conf` file using the testparm program. If testparm runs OK then it will list the loaded services. If not it will give an error message.

Make sure it runs OK and that the services look resonable before proceeding.

Step 5: Starting the smbd and nmbd

You must choose to start smbd and nmbd either as daemons or from **inetd**. Don't try to do both! Either you can put them in `inetd.conf` and have them started on demand by **inetd**, or you can start them as daemons either from the command line or in `/etc/rc.local`. See the man pages for details on the command line options. Take particular care to read the bit about what user you need to be in order to start Samba. In many cases you must be root.

The main advantage of starting **smbd** and **nmbd** as a daemon is that they will respond slightly more quickly to an initial connection request. This is, however, unlikely to be a problem.

Step 5a: Starting from inetd.conf

NOTE; The following will be different if you use NIS or NIS+ to distributed services maps.

Look at your `/etc/services`. What is defined at port 139/tcp. If nothing is defined then add a line like this:

```
netbios-ssn  139/tcp
```

similarly for 137/udp you should have an entry like:

```
netbios-ns 137/udp
```

Next edit your `/etc/inetd.conf` and add two lines something like this:

```
netbios-ssn stream tcp nowait root /usr/local/samba/bin/smbd smbd
netbios-ns dgram udp wait root /usr/local/samba/bin/nmbd nmbd
```

The exact syntax of `/etc/inetd.conf` varies between unixes. Look at the other entries in inetd.conf for a guide.

NOTE: Some unixes already have entries like netbios_ns (note the underscore) in `/etc/services`. You must either edit `/etc/services` or `/etc/inetd.conf` to make them consistant.

NOTE: On many systems you may need to use the "interfaces" option in smb.conf to specify the IP address and netmask of your interfaces. Run **ifconfig** as root if you don't know what the broadcast is for your net. **nmbd** tries to determine it at run time, but fails on somunixes. See the section on "testing nmbd" for a method of finding if you need to do this.

!!!WARNING!!! Many unixes only accept around 5 parameters on the command line in `inetd.conf`. This means you shouldn't use spaces between the options and arguments, or you should use a script, and start the script from **inetd**.

Restart **inetd**, perhaps just send it a HUP. If you have installed an earlier version of **nmbd** then you may need to kill nmbd as well.

Step 5b. Alternative: starting it as a daemon

To start the server as a daemon you should create a script something like this one, perhaps calling it `startsmb`.

```
#!/bin/sh
/usr/local/samba/bin/smbd -D
/usr/local/samba/bin/nmbd -D
```

then make it executable with **chmod +x startsmb**

You can then run **startsmb** by hand or execute it from `/etc/rc.local`

To kill it send a kill signal to the processes **nmbd** and **smbd**.

NOTE: If you use the SVR4 style init system then you may like to look at the `examples/svr4-startup` script to make Samba fit into that system.

Step 6: Try listing the shares available on your server

```
$ smbclient -L  yourhostname
```

Your should get back a list of shares available on your server. If you don't then something is incorrectly setup. Note that this method can also be used to see what shares are available on other LanManager clients (such as WfWg).

If you choose user level security then you may find that Samba requests a password before it will list the shares. See the **smbclient** man page for details. (you can force it to list the shares without a password by adding the option -U% to the command line. This will not work with non-Samba servers)

Step 7: Try connecting with the unix client

```
$ smbclient  //yourhostname/aservice
```

Typically the *yourhostname* would be the name of the host where you installed **smbd**. The *aservice* is any service you have defined in the `smb.conf` file. Try your user name if you just have a [homes] section in `smb.conf`.

For example if your unix host is bambi and your login name is fred you would type:

```
$ smbclient //bambi/fred
```

Step 8: Try connecting from a DOS, WfWg, Win9x, WinNT, Win2k, OS/2, etc... client

Try mounting disks. eg:

```
C:\WINDOWS\> net use d: \\servername\service
```

Try printing. eg:

```
C:\WINDOWS\> net use lpt1:  \\servername\spoolservice
```

```
C:\WINDOWS\> print filename
```

Celebrate, or send me a bug report!

What If Things Don't Work?

If nothing works and you start to think "who wrote this pile of trash" then I suggest you do step 2 again (and again) till you calm down.

Then you might read the file DIAGNOSIS.txt and the FAQ. If you are still stuck then try the mailing list or newsgroup (look in the README for details). Samba has been successfully installed at thousands of sites worldwide, so maybe someone else has hit your problem and has overcome it. You could also use the WWW site to scan back issues of the samba−digest.

When you fix the problem PLEASE send me some updates to the documentation (or source code) so that the next person will find it easier.

Diagnosing Problems

If you have instalation problems then go to `DIAGNOSIS.txt` to try to find the problem.

Scope IDs

By default Samba uses a blank scope ID. This means all your windows boxes must also have a blank scope ID. If you really want to use a non−blank scope ID then you will need to use the −i <scope> option to nmbd, smbd, and smbclient. All your PCs will need to have the same setting for this to work. I do not recommend scope IDs.

Choosing the Protocol Level

The SMB protocol has many dialects. Currently Samba supports 5, called CORE, COREPLUS, LANMAN1, LANMAN2 and NT1.

You can choose what maximum protocol to support in the `smb.conf` file. The default is NT1 and that is the best for the vast majority of sites.

In older versions of Samba you may have found it necessary to use COREPLUS. The limitations that led to this have mostly been fixed. It is now less likely that you will want to use less than LANMAN1. The only remaining advantage of COREPLUS is that for some obscure reason WfWg preserves the case of passwords in this protocol, whereas under LANMAN1, LANMAN2 or NT1 it uppercases all passwords before sending them, forcing you to use the "password level=" option in some cases.

The main advantage of LANMAN2 and NT1 is support for long filenames with some clients (eg: smbclient, Windows NT or Win95).

See the smb.conf(5) manual page for more details.

Note: To support print queue reporting you may find that you have to use TCP/IP as the default protocol under WfWg. For some reason if you leave Netbeui as the default it may break the print queue reporting on

some systems. It is presumably a WfWg bug.

Printing from UNIX to a Client PC

To use a printer that is available via a smb−based server from a unix host you will need to compile the smbclient program. You then need to install the script "smbprint". Read the instruction in smbprint for more details.

There is also a SYSV style script that does much the same thing called smbprint.sysv. It contains instructions.

Locking

One area which sometimes causes trouble is locking.

There are two types of locking which need to be performed by a SMB server. The first is "record locking" which allows a client to lock a range of bytes in a open file. The second is the "deny modes" that are specified when a file is open.

Samba supports "record locking" using the fcntl() unix system call. This is often implemented using rpc calls to a rpc.lockd process running on the system that owns the filesystem. Unfortunately many rpc.lockd implementations are very buggy, particularly when made to talk to versions from other vendors. It is not uncommon for the rpc.lockd to crash.

There is also a problem translating the 32 bit lock requests generated by PC clients to 31 bit requests supported by most unixes. Unfortunately many PC applications (typically OLE2 applications) use byte ranges with the top bit set as semaphore sets. Samba attempts translation to support these types of applications, and the translation has proved to be quite successful.

Strictly a SMB server should check for locks before every read and write call on a file. Unfortunately with the way fcntl() works this can be slow and may overstress the rpc.lockd. It is also almost always unnecessary as clients are supposed to independently make locking calls before reads and writes anyway if locking is important to them. By default Samba only makes locking calls when explicitly asked to by a client, but if you set "strict locking = yes" then it will make lock checking calls on every read and write.

You can also disable by range locking completely using "locking = no". This is useful for those shares that don't support locking or don't need it (such as cdroms). In this case Samba fakes the return codes of locking calls to tell clients that everything is OK.

The second class of locking is the "deny modes". These are set by an application when it opens a file to determine what types of access should be allowed simultaneously with its open. A client may ask for DENY_NONE, DENY_READ, DENY_WRITE or DENY_ALL. There are also special compatability modes called DENY_FCB and DENY_DOS.

You can disable share modes using "share modes = no". This may be useful on a heavily loaded server as the share modes code is very slow. See also the FAST_SHARE_MODES option in the Makefile for a way to do full share modes very fast using shared memory (if your OS supports it).

Mapping Usernames

If you have different usernames on the PCs and the unix server then take a look at the "username map" option. See the smb.conf man page for details.

Other Character Sets

If you have problems using filenames with accented characters in them (like the German, French or Scandinavian character sets) then I recommmend you look at the "valid chars" option in smb.conf and also take a look at the validchars package in the examples directory.

Chapter 2. LanMan and NT Password Encryption in Samba 2.x

Introduction

With the development of LanManager and Windows NT compatible password encryption for Samba, it is now able to validate user connections in exactly the same way as a LanManager or Windows NT server.

This document describes how the SMB password encryption algorithm works and what issues there are in choosing whether you want to use it. You should read it carefully, especially the part about security and the "PROS and CONS" section.

How does it work?

LanManager encryption is somewhat similar to UNIX password encryption. The server uses a file containing a hashed value of a user's password. This is created by taking the user's plaintext password, capitalising it, and either truncating to 14 bytes or padding to 14 bytes with null bytes. This 14 byte value is used as two 56 bit DES keys to encrypt a 'magic' eight byte value, forming a 16 byte value which is stored by the server and client. Let this value be known as the "hashed password".

Windows NT encryption is a higher quality mechanism, consisting of doing an MD4 hash on a Unicode version of the user's password. This also produces a 16 byte hash value that is non−reversible.

When a client (LanManager, Windows for WorkGroups, Windows 95 or Windows NT) wishes to mount a Samba drive (or use a Samba resource), it first requests a connection and negotiates the protocol that the client and server will use. In the reply to this request the Samba server generates and appends an 8 byte, random value − this is stored in the Samba server after the reply is sent and is known as the "challenge". The challenge is different for every client connection.

The client then uses the hashed password (16 byte values described above), appended with 5 null bytes, as three 56 bit DES keys, each of which is used to encrypt the challenge 8 byte value, forming a 24 byte value known as the "response".

In the SMB call SMBsessionsetupX (when user level security is selected) or the call SMBtconX (when share level security is selected), the 24 byte response is returned by the client to the Samba server. For Windows NT protocol levels the above calculation is done on both hashes of the user's password and both responses are returned in the SMB call, giving two 24 byte values.

The Samba server then reproduces the above calculation, using its own stored value of the 16 byte hashed password (read from the `smbpasswd` file − described later) and the challenge value that it kept from the negotiate protocol reply. It then checks to see if the 24 byte value it calculates matches the 24 byte value returned to it from the client.

If these values match exactly, then the client knew the correct password (or the 16 byte hashed value − see security note below) and is thus allowed access. If not, then the client did not know the correct password and is denied access.

Note that the Samba server never knows or stores the cleartext of the user's password − just the 16 byte hashed values derived from it. Also note that the cleartext password or 16 byte hashed values are never transmitted over the network − thus increasing security.

Important Notes About Security

The unix and SMB password encryption techniques seem similar on the surface. This similarity is, however, only skin deep. The unix scheme typically sends clear text passwords over the nextwork when logging in. This is bad. The SMB encryption scheme never sends the cleartext password over the network but it does store the 16 byte hashed values on disk. This is also bad. Why? Because the 16 byte hashed values are a "password equivalent". You cannot derive the user's password from them, but they could potentially be used in a modified client to gain access to a server. This would require considerable technical knowledge on behalf of the attacker but is perfectly possible. You should thus treat the smbpasswd file as though it contained the cleartext passwords of all your users. Its contents must be kept secret, and the file should be protected accordingly.

Ideally we would like a password scheme which neither requires plain text passwords on the net or on disk. Unfortunately this is not available as Samba is stuck with being compatible with other SMB systems (WinNT, WfWg, Win95 etc).

Warning
Note that Windows NT 4.0 Service pack 3 changed the default for permissible authentication so that plaintext passwords are *never* sent over the wire. The solution to this is either to switch to encrypted passwords with Samba or edit the Windows NT registry to re−enable plaintext passwords. See the document WinNT.txt for details on how to do this. Other Microsoft operating systems which also exhibit this behavior includes • MS DOS Network client 3.0 with the basic network redirector installed • Windows 95 with the network redirector update installed • Windows 98 [se] • Windows 2000 *Note :*All current release of Microsoft SMB/CIFS clients support authentication via the SMB Challenge/Response mechanism described here. Enabling clear text authentication does not disable the ability of the client to particpate in encrypted authentication.

Advantages of SMB Encryption

- plain text passwords are not passed across the network. Someone using a network sniffer cannot just record passwords going to the SMB server.
-

WinNT doesn't like talking to a server that isn't using SMB encrypted passwords. It will refuse to browse the server if the server is also in user level security mode. It will insist on prompting the user for the password on each connection, which is very annoying. The only things you can do to stop this is to use SMB encryption.

Advantages of non−encrypted passwords

- plain text passwords are not kept on disk.
- uses same password file as other unix services such as login and ftp
- you are probably already using other services (such as telnet and ftp) which send plain text passwords over the net, so sending them for SMB isn't such a big deal.

The smbpasswd file

In order for Samba to participate in the above protocol it must be able to look up the 16 byte hashed values given a user name. Unfortunately, as the UNIX password value is also a one way hash function (ie. it is impossible to retrieve the cleartext of the user's password given the UNIX hash of it), a separate password file containing this 16 byte value must be kept. To minimise problems with these two password files, getting out of sync, the UNIX `/etc/passwd` and the `smbpasswd` file, a utility, **mksmbpasswd.sh**, is provided to generate a smbpasswd file from a UNIX `/etc/passwd` file.

To generate the smbpasswd file from your `/etc/passwd` file use the following command :

```
$ cat /etc/passwd | mksmbpasswd.sh  > /usr/local/samba/private/smbpasswd
```

If you are running on a system that uses NIS, use

```
$ ypcat passwd | mksmbpasswd.sh  > /usr/local/samba/private/smbpasswd
```

The **mksmbpasswd.sh** program is found in the Samba source directory. By default, the smbpasswd file is stored in :

```
/usr/local/samba/private/smbpasswd
```

The owner of the `/usr/local/samba/private/` directory should be set to root, and the permissions on it should be set to 0500 (**chmod 500 /usr/local/samba/private**).

Likewise, the smbpasswd file inside the private directory should be owned by root and the permissions on is should be set to 0600 (**chmod 600 smbpasswd**).

The format of the smbpasswd file is (The line has been wrapped here. It should appear as one entry per line in your smbpasswd file.)

```
username:uid:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX:
        [Account type]:LCT-<last-change-time>:Long name
```

Although only the *username*, *uid*, *XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*, [*Account type*] and *last-change-time* sections are significant and are looked at in the Samba code.

It is *VITALLY* important that there by 32 'X' characters between the two ':' characters in the XXX sections − the smbpasswd and Samba code will fail to validate any entries that do not have 32 characters between ':' characters. The first XXX section is for the Lanman password hash, the second is for the Windows NT version.

When the password file is created all users have password entries consisting of 32 'X' characters. By default this disallows any access as this user. When a user has a password set, the 'X' characters change to 32 ascii hexadecimal digits (0−9, A−F). These are an ascii representation of the 16 byte hashed value of a user's password.

To set a user to have no password (not recommended), edit the file using vi, and replace the first 11 characters with the ascii text `"NO PASSWORD"` (minus the quotes).

For example, to clear the password for user bob, his smbpasswd file entry would look like :

```
bob:100:NO PASSWORDXXXXXXXXXXXXXXXXXXXXX:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX:[U          ]:L
```

If you are allowing users to use the smbpasswd command to set their own passwords, you may want to give users NO PASSWORD initially so they do not have to enter a previous password when changing to their new password (not recommended). In order for you to allow this the **smbpasswd** program must be able to connect to the **smbd** daemon as that user with no password. Enable this by adding the line :

null passwords = yes

to the [global] section of the smb.conf file (this is why the above scenario is not recommended). Preferably, allocate your users a default password to begin with, so you do not have to enable this on your server.

Note : This file should be protected very carefully. Anyone with access to this file can (with enough knowledge of the protocols) gain access to your SMB server. The file is thus more sensitive than a normal unix `/etc/passwd` file.

The smbpasswd Command

The smbpasswd command maintains the two 32 byte password fields in the smbpasswd file. If you wish to make it similar to the unix **passwd** or **yppasswd** programs, install it in `/usr/local/samba/bin/` (or your main Samba binary directory).

Note that as of Samba 1.9.18p4 this program *MUST NOT BE INSTALLED* setuid root (the new **smbpasswd** code enforces this restriction so it cannot be run this way by accident).

smbpasswd now works in a client−server mode where it contacts the local smbd to change the user's password on its behalf. This has enormous benefits − as follows.

- smbpasswd no longer has to be setuid root − an enormous range of potential security problems is eliminated.
- **smbpasswd** now has the capability to change passwords on Windows NT servers (this only works when the request is sent to the NT Primary Domain Controller if you are changing an NT Domain user's password).

To run smbpasswd as a normal user just type :

```
$ smbpasswd

Old SMB password: <type old value here − or hit return if there was no
old password>

New SMB Password: <type new value>

Repeat New SMB Password: <re−type new value
```

If the old value does not match the current value stored for that user, or the two new values do not match each other, then the password will not be changed.

If invoked by an ordinary user it will only allow the user to change his or her own Samba password.

If run by the root user smbpasswd may take an optional argument, specifying the user name whose SMB password you wish to change. Note that when run as root smbpasswd does not prompt for or check the old password value, thus allowing root to set passwords for users who have forgotten their passwords.

smbpasswd is designed to work in the same way and be familiar to UNIX users who use the **passwd** or **yppasswd** commands.

For more details on using **smbpasswd** refer to the man page which will always be the definitive reference.

Setting up Samba to support LanManager Encryption

This is a very brief description on how to setup samba to support password encryption.

1. compile and install samba as usual

2. enable encrypted passwords in `smb.conf` by adding the line **encrypt passwords = yes** in the [global] section

3. create the initial `smbpasswd` password file in the place you specified in the Makefile (−−prefix=<dir>). See the notes under the The smbpasswd File section earlier in the document for details.

Note that you can test things using smbclient.

Chapter 3. Hosting a Microsoft Distributed File System tree on Samba

Instructions

The Distributed File System (or Dfs) provides a means of separating the logical view of files and directories that users see from the actual physical locations of these resources on the network. It allows for higher availability, smoother storage expansion, load balancing etc. For more information about Dfs, refer to Microsoft documentation.

This document explains how to host a Dfs tree on a Unix machine (for Dfs−aware clients to browse) using Samba.

A Samba server can be made a Dfs server by setting the global boolean *host msdfs* parameter in the `smb.conf` file. You designate a share as a Dfs root using the share level boolean *msdfs root* parameter. A Dfs root directory on Samba hosts Dfs links in the form of symbolic links that point to other servers. For example, a symbolic link `junction−>msdfs:storage1share1` in the share directory acts as the Dfs junction. When Dfs−aware clients attempt to access the junction link, they are redirected to the storage location (in this case, \\storage1\share1).

Dfs trees on Samba work with all Dfs−aware clients ranging from Windows 95 to 2000.

Here's an example of setting up a Dfs tree on a Samba server.

```
# The smb.conf file:
[global]
        netbios name = SAMBA
        host msdfs   = yes

[dfs]
        path = /export/dfsroot
        msdfs root = yes
```

In the /export/dfsroot directory we set up our dfs links to other servers on the network.

```
root# cd /export/dfsroot

root# chown root /export/dfsroot

root# chmod 755 /export/dfsroot

root# ln −s msdfs:storageA\\shareA linka

root# ln −s msdfs:serverB\\share,serverC\\share linkb
```

You should set up the permissions and ownership of the directory acting as the Dfs root such that only designated users can create, delete or modify the msdfs links. Also note that symlink names should be all lowercase. This limitation exists to have Samba avoid trying all the case combinations to get at the link name. Finally set up the symbolic links to point to the network shares you want, and start Samba.

Users on Dfs−aware clients can now browse the Dfs tree on the Samba server at \\samba\dfs. Accessing links linka or linkb (which appear as directories to the client) takes users directly to the appropriate shares on the network.

Notes

- Windows clients need to be rebooted if a previously mounted non−dfs share is made a dfs root or vice versa. A better way is to introduce a new share and make it the dfs root.
- Currently there's a restriction that msdfs symlink names should all be lowercase.
- For security purposes, the directory acting as the root of the Dfs tree should have ownership and permissions set so that only designated users can modify the symbolic links in the directory.

Chapter 4. Printing Support in Samba 2.2.x

Introduction

Beginning with the 2.2.0 release, Samba now supports the native Windows NT printing mechanisms implemented via MS−RPC (i.e. the SPOOLSS named pipe). Previous versions of Samba only supported the LanMan printing calls.

The additional functionality provided by the new SPOOLSS support includes:

- Support for downloading printer driver files to Windows 95/98/NT/2000 clients upon demand.
- Uploading of printer drivers via the Windows NT Add Printer Wizard (APW) or the Imprints tool set
- Support for the native MS−RPC printing calls such as StartDocPrinter, EnumJobs(), etc... (See the MSDN documentation for more information on the Win32 printing API)
- Support for NT Access Control Lists (ACL) on printer objects
- Improved support for printer queue manipulation through the use of an internal database for spooled job information

---

Configuration

In order to support the uploading of printer driver files, you must first configure a file share named [print$]. The name of this share is hard coded in Samba's internals so the name is very important (print$ is the service used by Windows NT print servers to provide support for printer driver download.

Warning
Previous versions of Samba recommended using a share named [printer$]. This name was taken from the printer$ service created by Windows 9x clients when a printer was shared. Windows 9x printer servers always have a printer$ service which provides read−only access via no password in order to support printer driver downloads. However, the initial implementation allowed for a parameter named *`printer driver location`* to be used on a per share basis to specify the location of the driver files associated with that printer. Another parameter named *`printer driver`* provided a means of defining the printer driver name to be sent to the client. These parameters, including *`printer driver file`* parameter, are being depreciated and should not be used in new installations. For more information on this change, you should refer to the Migration section of this document.

You should modify the server's smb.conf file to create the following file share (of course, some of the parameter values, such as 'path' are arbitrary and should be replaced with appropriate values for your site):

```
[print$]
        path = /usr/local/samba/printers
        guest ok = yes
        browseable = yes
        read only = yes
        write list = ntadmin
```

The *`write list`* is used to allow administrative level user accounts to have write access in order to update files on the share. See the smb.conf(5) man page for more information on configuring file shares.

The requirement for **guest ok = yes** depends upon how your site is configured. If users will be guaranteed to have an account on the Samba host, then this is a non−issue.

author's note: The non−issue is that if all your Windows NT users are guarenteed to be authenticated by the Samba server (such as a domain member server and the NT user has already been validated by the Domain Controller in order to logon to the Windows NT console), then guest access is not necessary. Of course, in a workgroup environment where you just want to be able to print without worrying about silly accounts and security, then configure the share for guest access. You'll probably want to add **map to guest = Bad User** in the [global] section as well. Make sure you understand what this parameter does before using it though. −−jerry]

In order for a Windows NT print server to support the downloading of driver files by multiple client architectures, it must create subdirectories within the [print$] service which correspond to each of the supported client architectures. Samba follows this model as well.

Next create the directory tree below the [print$] share for each architecture you wish to support.

```
[print$]-----
        |-W32X86                ; "Windows NT x86"
        |-WIN40                 ; "Windows 95/98"
        |-W32ALPHA              ; "Windows NT Alpha_AXP"
        |-W32MIPS               ; "Windows NT R4000"
        |-W32PPC                ; "Windows NT PowerPC"
```

Warning

ATTENTION! REQUIRED PERMISSIONS

In order to currently add a new driver to you Samba host, one of two conditions must hold true:

- The account used to connect to the Samba host must have a uid of 0 (i.e. a root account)
- The account used to connect to the Samba host must be a member of the *printer admin* list.

Of course, the connected account must still possess access to add files to the sibdirectories beneath [print$].

Once you have created the required [print$] service and associated subdirectories, simply log onto the Samba server using a root (or *printer admin*) account from a Windows NT 4.0 client. Navigate to the "Printers" folder on the Samba server. You should see an initial listing of printers that matches the printer shares defined on your Samba host.

It is possible on a Windows NT print server to have printers listed in the Printers folder which are not shared. Samba does not make this distinction. By definition, the only printers of which Samba is aware are those which are specified as shares in `smb.conf`.

Another interesting side note is that Windows NT clients do not use the SMB printer share, but rather can print directly to any printer on another Windows NT host using MS−RPC. This of course assumes that the printing client has the necessary privileges on the remote host serving the printer. The default permissions assigned by Windows NT to a printer gives the "Print" permissions to the "Everyone" well−known group.

The initial listing of printers in the Samba host's Printers folder will have no printer driver assigned to them. The way assign a driver to a printer is to view the Properties of the printer and either

- Use the "New Driver..." button to install a new printer driver, or
- Select a driver from the popup list of installed drivers. Initially this list will be empty.

If you wish to install printer drivers for client operating systems other than "Windows NT x86", you will need to use the "Sharing" tab of the printer properties dialog.

Assuming you have connected with a root account, you will also be able modify other printer properties such

as ACLs and device settings using this dialog box.

The Imprints Toolset

The Imprints tool set provides a UNIX equivalent of the Windows NT Add Printer Wizard. For complete information, please refer to the Imprints web site at http://imprints.sourceforge.net/ as well as the documentation included with the imprints source distribution. This section will only provide a brief introduction to the features of Imprints.

What is Imprints?

Imprints is a collection of tools for supporting the goals of

- Providing a central repository information regarding Windows NT and 95/98 printer driver packages
- Providing the tools necessary for creating the Imprints printer driver packages.
- Providing an installation client which will obtain and install printer drivers on remote Samba and Windows NT 4 print servers.

Creating Printer Driver Packages

The process of creating printer driver packages is beyond the scope of this document (refer to Imprints.txt also included with the Samba distribution for more information). In short, an Imprints driver package is a gzipped tarball containing the driver files, related INF files, and a control file needed by the installation client.

The Imprints server

The Imprints server is really a database server that may be queried via standard HTTP mechanisms. Each printer entry in the database has an associated URL for the actual downloading of the package. Each package is digitally signed via GnuPG which can be used to verify that package downloaded is actually the one referred in the Imprints database. It is *not* recommended that this security check be disabled.

The Installation Client

More information regarding the Imprints installation client is available in the `Imprints-Client-HOWTO.ps` file included with the imprints source package.

The Imprints installation client comes in two forms.

- a set of command line Perl scripts
-

a GTK+ based graphical interface to the command line perl scripts

The installation client (in both forms) provides a means of querying the Imprints database server for a matching list of known printer model names as well as a means to download and install the drivers on remote Samba and Windows NT print servers.

The basic installation process is in four steps and perl code is wrapped around **smbclient** and **rpcclient**.

```
foreach (supported architecture for a given driver)
{
        1.      rpcclient: Get the appropriate upload directory
                on the remote server
        2.      smbclient: Upload the driver files
        3.      rpcclient: Issues an AddPrinterDriver() MS-RPC
}

4.      rpcclient: Issue an AddPrinterEx() MS-RPC to actually
        create the printer
```

One of the problems encountered when implementing the Imprints tool set was the name space issues between various supported client architectures. For example, Windows NT includes a driver named "Apple LaserWriter II NTX v51.8" and Windows 95 callsits version of this driver "Apple LaserWriter II NTX"

The problem is how to know what client drivers have been uploaded for a printer. As astute reader will remember that the Windows NT Printer Properties dialog only includes space for one printer driver name. A quick look in the Windows NT 4.0 system registry at

`HKLM\System\CurrentControlSet\Control\Print\Environment`

will reveal that Windows NT always uses the NT driver name. The is ok as Windows NT always requires that at least the Windows NT version of the printer driver is present. However, Samba does not have the requirement internally. Therefore, how can you use the NT driver name if is has not already been installed?

The way of sidestepping this limitation is to require that all Imprints printer driver packages include both the Intel Windows NT and 95/98 printer drivers and that NT driver is installed first.

Migration to from Samba 2.0.x to 2.2.x

Given that printer driver management has changed (we hope improved :)) in 2.2.0 over prior releases, migration from an existing setup to 2.2.0 can follow several paths.

Warning
The following smb.conf parameters are considered to be depreciated and will be removed soon. Do not use them in new installations • *printer driver file (G)* • *printer driver (S)* • *printer driver location (S)*

Here are the possible scenarios for supporting migration:

- If you does not desire the new Windows NT print driver support, nothing needs to be done. All existing parameters work the same.
- If you want to take advantage of NT printer driver support but does not want to migrate the 9x drivers to the new setup, the leave the existing printers.def file. When smbd attempts to locate a 9x driver for the printer in the TDB and fails it will drop down to using the printers.def (and all associated parameters). The **make_printerdef** tool will also remain for backwards compatibility but will be moved to the "this tool is the old way of doing it" pile.
- If you install a Windows 9x driver for a printer on your Samba host (in the printing TDB), this information will take precedence and the three old printing parameters will be ignored (including print driver location).
- If you want to migrate an existing `printers.def` file into the new setup, the current only solution is to use the Windows NT APW to install the NT drivers and the 9x drivers. (comment: this could possibly be scripted using smbclient and rpcclient, but I haven't had time --jerry)

Chapter 5. security = domain in Samba 2.x

Joining an NT Domain with Samba 2.2

In order for a Samba−2 server to join an NT domain, you must first add the NetBIOS name of the Samba server to the NT domain on the PDC using Server Manager for Domains. This creates the machine account in the domain (PDC) SAM. Note that you should add the Samba server as a "Windows NT Workstation or Server", *NOT* as a Primary or backup domain controller.

Assume you have a Samba−2 server with a NetBIOS name of `SERV1` and are joining an NT domain called `DOM`, which has a PDC with a NetBIOS name of `DOMPDC` and two backup domain controllers with NetBIOS names `DOMBDC1` and `DOMBDC2` .

In order to join the domain, first stop all Samba daemons and run the command:

```
root# smbpasswd −j DOM −r DOMPDC
```

as we are joining the domain DOM and the PDC for that domain (the only machine that has write access to the domain SAM database) is DOMPDC. If this is successful you will see the message:

```
smbpasswd: Joined domain DOM.
```

in your terminal window. See the smbpasswd(8) man page for more details.

This command goes through the machine account password change protocol, then writes the new (random) machine account password for this Samba server into a file in the same directory in which an smbpasswd file would be stored − normally :

```
/usr/local/samba/private
```

In Samba 2.0.x, the filename looks like this:

```
<NT DOMAIN NAME>.<Samba Server Name>.mac
```

The `.mac` suffix stands for machine account password file. So in our example above, the file would be called:

```
DOM.SERV1.mac
```

In Samba 2.2, this file has been replaced with a TDB (Trivial Database) file named `secrets.tdb`.

This file is created and owned by root and is not readable by any other user. It is the key to the domain−level security for your system, and should be treated as carefully as a shadow password file.

Now, before restarting the Samba daemons you must edit your `smb.conf(5)` file to tell Samba it should now use domain security.

Change (or add) your *`security =`* line in the [global] section of your smb.conf to read:

security = domain

Next change the *`workgroup =`* line in the [global] section to read:

workgroup = DOM

as this is the name of the domain we are joining.

You must also have the parameter *encrypt passwords* set to `yes` in order for your users to authenticate to the NT PDC.

Finally, add (or modify) a *password server =* line in the [global] section to read:

password server = DOMPDC DOMBDC1 DOMBDC2

These are the primary and backup domain controllers Samba will attempt to contact in order to authenticate users. Samba will try to contact each of these servers in order, so you may want to rearrange this list in order to spread out the authentication load among domain controllers.

Alternatively, if you want smbd to automatically determine the list of Domain controllers to use for authentication, you may set this line to be :

password server = *

This method, which was introduced in Samba 2.0.6, allows Samba to use exactly the same mechanism that NT does. This method either broadcasts or uses a WINS database in order to find domain controllers to authenticate against.

Finally, restart your Samba daemons and get ready for clients to begin using domain security!

Why is this better than security = server?

Currently, domain security in Samba doesn't free you from having to create local Unix users to represent the users attaching to your server. This means that if domain user `DOM\fred` attaches to your domain security Samba server, there needs to be a local Unix user fred to represent that user in the Unix filesystem. This is very similar to the older Samba security mode security = server, where Samba would pass through the authentication request to a Windows NT server in the same way as a Windows 95 or Windows 98 server would.

The advantage to domain−level security is that the authentication in domain−level security is passed down the authenticated RPC channel in exactly the same way that an NT server would do it. This means Samba servers now participate in domain trust relationships in exactly the same way NT servers do (i.e., you can add Samba servers into a resource domain and have the authentication passed on from a resource domain PDC to an account domain PDC.

In addition, with **security = server** every Samba daemon on a server has to keep a connection open to the authenticating server for as long as that daemon lasts. This can drain the connection resources on a Microsoft NT server and cause it to run out of available connections. With **security = domain**, however, the Samba daemons connect to the PDC/BDC only for as long as is necessary to authenticate the user, and then drop the connection, thus conserving PDC connection resources.

And finally, acting in the same manner as an NT server authenticating to a PDC means that as part of the authentication reply, the Samba server gets the user identification information such as the user SID, the list of NT groups the user belongs to, etc. All this information will allow Samba to be extended in the future into a mode the developers currently call appliance mode. In this mode, no local Unix users will be necessary, and Samba will generate Unix uids and gids from the information passed back from the PDC when a user is authenticated, making a Samba server truly plug and play in an NT domain environment. Watch for this code soon.

NOTE: Much of the text of this document was first published in the Web magazine LinuxWorld as the article Doing the NIS/NT Samba.

Chapter 6. UNIX Permission Bits and WIndows NT Access Control Lists

Viewing and changing UNIX permissions using the NT security dialogs

New in the Samba 2.0.4 release is the ability for Windows NT clients to use their native security settings dialog box to view and modify the underlying UNIX permissions.

Note that this ability is careful not to compromise the security of the UNIX host Samba is running on, and still obeys all the file permission rules that a Samba administrator can set.

In Samba 2.0.4 and above the default value of the parameter *nt acl support* has been changed from `false` to `true`, so manipulation of permissions is turned on by default.

How to view file security on a Samba share

From an NT 4.0 client, single−click with the right mouse button on any file or directory in a Samba mounted drive letter or UNC path. When the menu pops−up, click on the *Properties* entry at the bottom of the menu. This brings up the normal file properties dialog box, but with Samba 2.0.4 this will have a new tab along the top marked *Security*. Click on this tab and you will see three buttons, *Permissions*, *Auditing*, and *Ownership*. The *Auditing* button will cause either an error message A requested privilege is not held by the client to appear if the user is not the NT Administrator, or a dialog which is intended to allow an Administrator to add auditing requirements to a file if the user is logged on as the NT Administrator. This dialog is non−functional with a Samba share at this time, as the only useful button, the **Add** button will not currently allow a list of users to be seen.

Viewing file ownership

Clicking on the **"Ownership"** button brings up a dialog box telling you who owns the given file. The owner name will be of the form :

"SERVER\user (Long name)"

Where *SERVER* is the NetBIOS name of the Samba server, *user* is the user name of the UNIX user who owns the file, and *(Long name)* is the discriptive string identifying the user (normally found in the GECOS field of the UNIX password database). Click on the **Close** button to remove this dialog.

If the parameter *nt acl support* is set to false then the file owner will be shown as the NT user **"Everyone"**.

The **Take Ownership** button will not allow you to change the ownership of this file to yourself (clicking on it will display a dialog box complaining that the user you are currently logged onto the NT client cannot be found). The reason for this is that changing the ownership of a file is a privilaged operation in UNIX, available only to the *root* user. As clicking on this button causes NT to attempt to change the ownership of a file to the current user logged into the NT client this will not work with Samba at this time.

There is an NT chown command that will work with Samba and allow a user with Administrator privillage connected to a Samba 2.0.4 server as root to change the ownership of files on both a local NTFS filesystem or remote mounted NTFS or Samba drive. This is available as part of the *Seclib* NT security library written by Jeremy Allison of the Samba Team, available from the main Samba ftp site.

Viewing file or directory permissions

The third button is the **"Permissions"** button. Clicking on this brings up a dialog box that shows both the permissions and the UNIX owner of the file or directory. The owner is displayed in the form :

"SERVER\user (Long name)"

Where *SERVER* is the NetBIOS name of the Samba server, *user* is the user name of the UNIX user who owns the file, and *(Long name)* is the discriptive string identifying the user (normally found in the GECOS field of the UNIX password database).

If the parameter *nt acl support* is set to false then the file owner will be shown as the NT user **"Everyone"** and the permissions will be shown as NT "Full Control".

The permissions field is displayed differently for files and directories, so I'll describe the way file permissions are displayed first.

File Permissions

The standard UNIX user/group/world triple and the correspinding "read", "write", "execute" permissions triples are mapped by Samba into a three element NT ACL with the 'r', 'w', and 'x' bits mapped into the corresponding NT permissions. The UNIX world permissions are mapped into the global NT group **Everyone**, followed by the list of permissions allowed for UNIX world. The UNIX owner and group permissions are displayed as an NT **user** icon and an NT **local group** icon respectively followed by the list of permissions allowed for the UNIX user and group.

As many UNIX permission sets don't map into common NT names such as **"read"**, **"change"** or **"full control"** then usually the permissions will be prefixed by the words **"Special Access"** in the NT display list.

But what happens if the file has no permissions allowed for a particular UNIX user group or world component ? In order to allow "no permissions" to be seen and modified then Samba overloads the NT **"Take Ownership"** ACL attribute (which has no meaning in UNIX) and reports a component with no permissions as having the NT **"O"** bit set. This was chosen of course to make it look like a zero, meaning zero permissions. More details on the decision behind this will be given below.

Directory Permissions

Directories on an NT NTFS file system have two different sets of permissions. The first set of permissions is the ACL set on the directory itself, this is usually displayed in the first set of parentheses in the normal **"RW"** NT style. This first set of permissions is created by Samba in exactly the same way as normal file permissions are, described above, and is displayed in the same way.

The second set of directory permissions has no real meaning in the UNIX permissions world and represents the **"inherited"** permissions that any file created within this directory would inherit.

Samba synthesises these inherited permissions for NT by returning as an NT ACL the UNIX permission

mode that a new file created by Samba on this share would receive.

Modifying file or directory permissions

Modifying file and directory permissions is as simple as changing the displayed permissions in the dialog box, and clicking the **OK** button. However, there are limitations that a user needs to be aware of, and also interactions with the standard Samba permission masks and mapping of DOS attributes that need to also be taken into account.

If the parameter *nt acl support* is set to false then any attempt to set security permissions will fail with an **"Access Denied"** message.

The first thing to note is that the **"Add"** button will not return a list of users in Samba 2.0.4 (it will give an error message of **"The remote proceedure call failed and did not execute"**). This means that you can only manipulate the current user/group/world permissions listed in the dialog box. This actually works quite well as these are the only permissions that UNIX actually has.

If a permission triple (either user, group, or world) is removed from the list of permissions in the NT dialog box, then when the **"OK"** button is pressed it will be applied as "no permissions" on the UNIX side. If you then view the permissions again the "no permissions" entry will appear as the NT **"O"** flag, as described above. This allows you to add permissions back to a file or directory once you have removed them from a triple component.

As UNIX supports only the "r", "w" and "x" bits of an NT ACL then if other NT security attributes such as "Delete access" are selected then they will be ignored when applied on the Samba server.

When setting permissions on a directory the second set of permissions (in the second set of parentheses) is by default applied to all files within that directory. If this is not what you want you must uncheck the **"Replace permissions on existing files"** checkbox in the NT dialog before clicking **"OK"**.

If you wish to remove all permissions from a user/group/world component then you may either highlight the component and click the **"Remove"** button, or set the component to only have the special **"Take Ownership"** permission (dsplayed as **"O"**) highlighted.

Interaction with the standard Samba create mask parameters

Note that with Samba 2.0.5 there are four new parameters to control this interaction. These are :

security mask

force security mode

directory security mask

force directory security mode

Once a user clicks **"OK"** to apply the permissions Samba maps the given permissions into a user/group/world r/w/x triple set, and then will check the changed permissions for a file against the bits set in the *security mask* parameter. Any bits that were changed that are not set to '1' in this parameter are left alone in the file permissions.

Essentially, zero bits in the *security mask* mask may be treated as a set of bits the user is *not* allowed to change, and one bits are those the user is allowed to change.

If not set explicitly this parameter is set to the same value as the *create mask* parameter to provide compatibility with Samba 2.0.4 where this permission change facility was introduced. To allow a user to modify all the user/group/world permissions on a file, set this parameter to 0777.

Next Samba checks the changed permissions for a file against the bits set in the *force security mode* parameter. Any bits that were changed that correspond to bits set to '1' in this parameter are forced to be set.

Essentially, bits set in the *force security mode* parameter may be treated as a set of bits that, when modifying security on a file, the user has always set to be 'on'.

If not set explicitly this parameter is set to the same value as the *force create mode* parameter to provide compatibility with Samba 2.0.4 where the permission change facility was introduced. To allow a user to modify all the user/group/world permissions on a file, with no restrictions set this parameter to 000.

The *security mask* and *force security mode* parameters are applied to the change request in that order.

For a directory Samba will perform the same operations as described above for a file except using the parameter *directory security mask* instead of *security mask*, and *force directory security mode* parameter instead of *force security mode* .

The *directory security mask* parameter by default is set to the same value as the *directory mask* parameter and the *force directory security mode* parameter by default is set to the same value as the *force directory mode* parameter to provide compatibility with Samba 2.0.4 where the permission change facility was introduced.

In this way Samba enforces the permission restrictions that an administrator can set on a Samba share, whilst still allowing users to modify the permission bits within that restriction.

If you want to set up a share that allows users full control in modifying the permission bits on their files and directories and doesn't force any particular bits to be set 'on', then set the following parameters in the smb.conf(5) file in that share specific section :

```
security mask = 0777

force security mode = 0

directory security mask = 0777

force directory security mode = 0
```

As described, in Samba 2.0.4 the parameters :

```
create mask

force create mode

directory mask

force directory mode
```

were used instead of the parameters discussed here.

Interaction with the standard Samba file attribute mapping

Samba maps some of the DOS attribute bits (such as "read only") into the UNIX permissions of a file. This means there can be a conflict between the permission bits set via the security dialog and the permission bits set by the file attribute mapping.

One way this can show up is if a file has no UNIX read access for the owner it will show up as "read only" in the standard file attributes tabbed dialog. Unfortunately this dialog is the same one that contains the security info in another tab.

What this can mean is that if the owner changes the permissions to allow themselves read access using the security dialog, clicks **"OK"** to get back to the standard attributes tab dialog, and then clicks **"OK"** on that dialog, then NT will set the file permissions back to read−only (as that is what the attributes still say in the dialog). This means that after setting permissions and clicking **"OK"** to get back to the attributes dialog you should always hit **"Cancel"** rather than **"OK"** to ensure that your changes are not overridden.