,ch06.27287 Page 155 Friday, November 19, 1999 3:29 PM

6

Users, Security, and Domains

This chapter discusses how to configure users with the Samba server. This topic may seem straightforward at first, but you'll soon discover that there are several ancillary problems that can crop up. One issue that Samba administrators have difficulty with is user authentication—password and security problems are by far the most common support questions on the Samba mailing lists. Learning why various authentication mechanisms work on certain architectures (and don't on others) can save you a tremendous amount of time testing and debugging Samba users in the future.

Users and Groups

Before we start, we need to warn you up front that if you are connecting to Samba with a Windows 98 or NT 4.0 Workstation SP3, you need to configure your server for encrypted passwords before you can make a connection; otherwise, the clients will refuse to connect to the Samba server. This is because each of those Windows clients sends encrypted passwords, and Samba needs to be configured to expect and decrypt them. We'll show you how to set up Samba for this task later in the chapter, assuming you haven't already tackled this problem in Chapter 2, *Installing Samba on a Unix System.*

Let's start with a single user. The easiest way to set up a client user is to create a Unix account (and home directory) for that individual on the server, and notify Samba of the user's existence. You can do the latter by creating a disk share that maps to the user's home directory in the Samba configuration file, and restricting access to that user with the `valid users` option. For example:

```
[dave]
        path = /home/dave
        comment = Dave's home directory
        writeable = yes
        valid users = dave
```

The `valid users` option lists the users that will be allowed to access the share. In this case, only the user `dave` is allowed to access the share. In the previous chapters, we specified that any user could access a disk share using the `guest ok` parameter. Because we don't wish to allow guest access, that option is absent here. We could grant both authenticated users and guest users access to a specific share if we wanted to. The difference between the two typically involves access rights for each of the files.

Remember that you can abbreviate the user's home directory by using the `%H` variable. In addition, you can use the Unix username variable `%u` and/or the client username variable `%U` in your options as well. For example:

```
[dave]
    comment = %U home directory
    writeable = yes
    valid users = dave
    path = %H
```

Both of these examples work as long as the Unix user that Samba uses to represent the client has read/write access to the directory referenced by the `path` option. In other words, a client must first pass Samba's security mechanisms (e.g., encrypted passwords, the `valid users` option, etc.) as well as the normal Unix file and directory permissions of its Unix-side user *before* it can gain read/write access to a share.

With a single user accessing a home directory, access permissions are taken care of when the operating system creates the user account. However, if you're creating a shared directory for group access, there are a few more steps you need to perform. Let's take a stab at a group share for the accounting department in the *smb.conf* file:

```
[accounting]
    comment = Accounting Department Directory
    writeable = yes
    valid users = @account
    path = /home/samba/accounting
    create mode = 0660
    directory mode = 0770
```

The first thing that you might notice we did differently is to specify `@account` as the valid user instead of one or more individual usernames. This is shorthand for saying that the valid users are represented by the Unix group `account`. These users will need to be added to the group entry `account` in the system group file (*/etc/group* or equivalent) to be recognized as part of the group. Once they are, Samba will recognize those users as valid users for the share.

In addition, you will need to create a shared directory that the members of the group can access, which is pointed to by the `path` configuration option. Here are the Unix commands that create the shared directory for the accounting department (assuming */home/samba* already exists):

```
# mkdir /home/samba/accounting
# chgrp account /home/samba/accounting
# chmod 770 /home/samba/accounting
```

There are two other options in this *smb.conf* example, both of which we saw in the previous chapter. These options are `create mode` and `directory mode`. These options set the maximum file and directory permissions that a new file or directory can have. In this case, we have denied all world access to the contents of this share. (This is reinforced by the *chmod* command, shown earlier.).

The [homes] Share

Let's return to user shares for a moment. If we have several users to set up home directory shares for, we probably want to use the special `[homes]` share that we introduced in Chapter 5, *Browsing and Advanced Disk Shares*. With the `[homes]` share, all we need to say is:

```
[homes]
    browsable = no
    writable = yes
```

The `[homes]` share is a special section of the Samba configuration file. If a user attempts to connect to an ordinary share that doesn't appear in the *smb.conf* file (such as specifying it with a UNC in Windows Explorer), Samba will search for a `[homes]` share. If one exists, the incoming share name is assumed to be a username and is queried as such in the password database (*/etc/passwd* or equivalent) file of the Samba server. If it appears, Samba assumes the client is a Unix user trying to connect to his or her home directory.

As an illustration, let's assume that `sofia` is attempting to connect to a share called `[sofia]` on the Samba server. There is no share by that name in the configuration file, but a `[homes]` share exists and user `sofia` is present in the password database, so Samba takes the following steps:

1. Samba creates a new disk share called `[sofia]` with the `path` specified in the `[homes]` section. If there is no `path` option specified in `[homes]`, Samba initializes it to her home directory.
2. Samba initializes the new share's options from the defaults in `[globals]`, and any overriding options in `[homes]` with the exception of `browseable`.
3. Samba connects `sofia`'s client to that share.

The [homes] share is a fast, painless way to create shares for your user community without having to duplicate the information from the password database file in the *smb.conf* file. It does have some peculiarities, however, that we need to point out:

- The [homes] section can represent any account on the machine, which isn't always desirable. For example, it can potentially create a share for *root*, *bin*, *sys*, *uucp*, and the like. (You can set a global **invalid users** option to protect against this.)
- The meaning of the **browseable** configuration option is different from other shares; it indicates only that a [homes] section won't show up in the local browse list, not that the [alice] share won't. When the [alice] section is created (after the initial connection), it will use the browsable value from the [globals] section for that share, not the value from [homes].

As we mentioned, there is no need for a path statement in [homes] if the users have Unix home directories in the server's */etc/passwd* file. You should ensure that a valid home directory does exist, however, as Samba will not automatically create a home directory for a user, and will refuse a tree connect if the user's directory does not exist or is not accessible.

Controlling Access to Shares

Often you will need to restrict the users who can access a specific share for security reasons. This is very easy to do with Samba since it contains a wealth of options for creating practically any security configuration. Let's introduce a few configurations that you might want to use in your own Samba setup.

Again, if you are connecting with Windows 98 or NT 4.0 with Service Pack 3 (or above), those clients will send encrypted passwords to the Samba server. If Samba is not configured for this, it will continually refuse the connection. This chapter describes how to set up Samba for encrypted passwords. See the "Passwords" section.

We've seen what happens when you specify valid users. However, you are also allowed to specify a list of invalid users—users who should never be allowed access to Samba or its shares. This is done with the **invalid users** option. We hinted at one frequent use of this option earlier: a global default with the [homes] section to ensure that various system users and superusers cannot be forged for access. For example:

```
[global]
    invalid users = root bin daemon adm sync shutdown \
                        halt mail news uucp operator gopher
    auto services = dave peter bob

[homes]
    browsable = no
    writeable = yes
```

The invalid users option, like valid users, can take group names as well as usernames. In the event that a user or group appears in both lists, the invalid users option takes precedence and the user or group will be denied access to the share.

At the other end of the spectrum, you can explicitly specify users who will be allowed superuser (root) access to a share with the admin users option. An example follows:

```
[sales]
        path = /home/sales
        comment = Fiction Corp Sales Data
        writeable = yes
        valid users = tom dick harry
        admin users = mike
```

This option takes both group names and usernames. In addition, you can specify NIS netgroups by preceding them with an @ as well; if the netgroup is not found, Samba will assume that you are referring to a standard Unix group.

Be careful if you assign an entire group administrative privileges to a share. The Samba team highly recommends you avoid using this option, as it essentially gives root access to the specified users or groups for that share.

If you wish to force read-only or read-write access to users who access a share, you can do so with the read list and write list options, respectively. These options can be used on a per-share basis to restrict a writable share or grant write access to specific users in a read-only share, respectively. For example:

```
[sales]
        path = /home/sales
        comment = Fiction Corp Sales Data
        read only = yes
        write list = tom dick
```

The write list option cannot override Unix permissions. If you've created the share without giving the write-list user write permission on the Unix system, he or she will be denied write access regardless of the setting of write list.

Guest Access

As mentioned earlier, you can specify users who have guest access to a share. The options that control guest access are easy to work with. The first option, guest account, specifies the Unix account that guest users should be assigned when connecting to the Samba server. The default value for this is set during compilation, and is typically nobody. However, you may want to reset the guest user to ftp if you have trouble accessing various system services.

If you wish to restrict access in a share only to guests—in other words, all clients connect as the guest account when accessing the share—you can use the guest only option in conjunction with the guest ok option, as shown in the following example:

```
[sales]
        path = /home/sales
        comment = Fiction Corp Sales Data
        writeable = yes
        guest ok = yes
        guest account = ftp
        guest only = yes
```

Make sure you specify yes for both guest only and guest ok in this scenario; otherwise, Samba will not use the guest acount that you specify.

Access Control Options

Table 6-1 summarizes the options that you can use to control access to shares.

Table 6-1. Share-level Access Options

Option	Parameters	Function	Default	Scope
admin users	string (list of usernames)	Specifies a list of users who can perform operations as root.	None	Share
valid users	string (list of usernames)	Specifies a list of users that can connect to a share.	None	Share
invalid users	string (list of usernames)	Specifies a list of users that will be denied access to a share.	None	Share
read list	string (list of usernames)	Specifies a list of users that have read-only access to a writable share.	None	Share
write list	string (list of usernames)	Specifies a list of users that have read-write access to a read-only share.	None	Share
max connections	numerical	Indicates the maximum number of connections for a share at a given time.	0	Share
guest only (only guest)	boolean	Specifies that this share allows only guest access.	no	Share
guest account	string (name of account)	Names the Unix account that will be used for guest access.	nobody	Share

admin users

This option specifies a list of users that perform file operations as if they were `root`. This means that they can modify or destroy any other user's work, no matter what the permissions. Any files that they create will have root ownership and will use the default group of the admin user. The `admin users` option is used to allow PC users to act as administrators for particular shares. We urge you to avoid this option.

valid users and invalid users

These two options let you enumerate the users and groups who are granted or denied access to a particular share. You can enter a list of comma-delimited users, or indicate an NIS or Unix group name by prefixing the name with an at-sign (`@`).

The important rule to remember with these options is that any name or group in the `invalid users` list will *always* be denied access, even if it is included (in any form) in the `valid users` list. By default, neither option has a value associated with it. If both options have no value, any user is allowed to access the share.

read list and write list

Like the `valid users and invalid users` options, this pair of options specifies which users have read-only access to a writeable share and read-write access to a read-only share, respectively. The value of either options is a list of users. `read list` overrides any other Samba permissions granted—as well as Unix file permissions on the server system—to deny users write access. `write list` overrides other Samba permissions to grant write access, but cannot grant write access if the user lacks write permissions for the file on the Unix system. You can specify NIS or Unix group names by prefixing the name with an at sign (such as `@users`). Neither configuration option has a default value associated with it.

max connections

This option specifies the maximum number of client connections that a share can have at any given time. Any connections that are attempted after the maximum is reached will be rejected. The default value is `0`, which means that an unlimited number of connections are allowed. You can override it per share as follows:

```
[accounting]
    max connections = 30
```

This option is useful in the event that you need to limit the number of users who are accessing a licensed program or piece of data concurrently.

guest only

This share-level option (sometimes called `only guest`) forces a connection to a share to be performed with the user specified by the `guest account` option. The share to which this is applied must explicitly specify `guest ok = yes` in order for this option to be recognized by Samba. The default value for this option is `no`.

guest account

This option specifies the name of account to be used for guest access to shares in Samba. The default for this option varies from system to system, but it is often set to `nobody`. Some default user accounts have trouble connecting as guest users. If that occurs on your system, the Samba team recommends using the ftp account as the guest user.

Username Options

Table 6-2 shows two additional options that Samba can use to correct for incompatibilities in usernames between Windows and Unix.

Table 6-2. Username Options

Option	Parameters	Function	Default	Scope
`username map`	string (fully-qualified pathname)	Sets the name of the username mapping file.	None	Global
`username level`	numerical	Indicates the number of capital letters to use when trying to match a username.	`0`	Global

username map

Client usernames on an SMB network can be relatively large (up to 255 characters), while usernames on a Unix network often cannot be larger than eight characters. This means that an individual user may have one username on a client and another (shorter) one on the Samba server. You can get past this issue by *mapping* a free-form client username to a Unix username of eight or fewer characters. It is placed in a standard text file, using a format that we'll describe shortly. You can then specify the pathname to Samba with the global `username map` option. Be sure to restrict access to this file; make the root user the file's owner and deny write access to others. Otherwise, an untrusted user who can access the file can easily map their client username to the root user of the Samba server.

You can specify this option as follows:

```
[global]
    username map = /etc/samba/usermap.txt
```

Each of the entries in the username map file should be listed as follows: the Unix username, followed by an equal sign (=), followed by one or more whitespace-separated SMB client usernames. Note that unless instructed otherwise, (i.e., a guest connection), Samba will expect both the client and the server user to have the same password. You can also map NT groups to one or more specific Unix groups using the `@` sign. Here are some examples:

```
jarwin = JosephArwin
manderso = MarkAnderson
users = @account
```

Also, you can use the asterisk to specify a wildcard that matches any free-form client username as an entry in the username map file:

```
nobody = *
```

Comments in the file can be specified as lines beginning with (`#`) and (`;`).

Note that you can also use this file to redirect one Unix user to another user. Be careful if you do so because Samba and your client may not notify the user that the mapping has been made and Samba may be expecting a different password.

username level

SMB clients (such as Windows) will often send usernames in SMB connection requests entirely in capital letters; in other words, client usernames are not necessarily case sensitive. On a Unix server, however, usernames *are* case sensitive: the user `ANDY` is different from the user `andy`. By default, Samba attacks this problem by doing the following:

1. Checking for a user account with the exact name sent by the client
2. Testing the username in all lowercase letters
3. Testing the username in lowercase letters with only the first letter capitalized

If you wish to have Samba attempt more combinations of uppercase and lowercase letters, you can use the `username level` global configuration option. This option takes an integer value that specifies how many letters in the username should be capitalized when attempting to connect to a share. You can specify this options as follows:

```
[global]
    username level = 3
```

In this case, Samba will then attempt all permutations of usernames it can compute having three capital letters. The larger the number, the more computations Samba will have to perform to match the username and the longer the authentication will take.

Authentication Security

At this point, we should discuss how Samba authenticates users. Each user who attempts to connect to a share that does not allow guest access must provide a password to make a successful connection. What Samba does with that password—and consequently the strategy Samba will use to handle user authentication—is the arena of the `security` configuration option. There are currently four security levels that Samba supports on its network: *share*, *user*, *server*, and *domain*.

Share-level security
: Each share in the workgroup has one or more passwords associated with it. Anyone who knows a valid password for the share can access it.

User-level security
: Each share in the workgroup is configured to allow access from certain users. With each initial tree connection, the Samba server verifies users and their passwords to allow them access to the share.

Server-level security
: This is the same as user-level security, except that the Samba server uses a separate SMB server to validate users and their passwords before granting access to the share.

Domain-level security
: Samba becomes a member of a Windows domain and uses the domain's primary domain controller (PDC) to perform authentication. Once authenticated, the user is given a special token that allows him or her access to any share with appropriate access rights. With this token, the PDC will not have to revalidate the user's password each time he or she attempts to access another share within the domain.

Each of these security policies can be implemented with the global `security` option, as shown in Table 6-3.

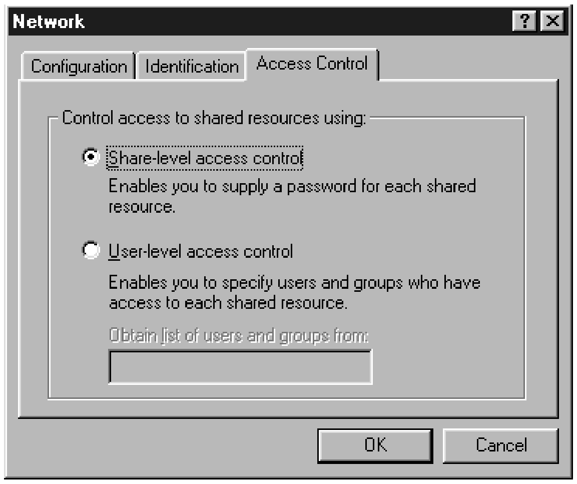
Table 6-3. Security Option

Option	Parameters	Function	Default	Scope
`security`	`domain`, `server`, `share`, or `user`	Indicates the type of security that the Samba server will use.	`user` (Samba 2.0) or `share` (Samba 1.9)	Global

Share-level Security

With share-level security, each share has one or more passwords associated with it. This differs from the other modes of security in that there are no restrictions as to whom can access a share, as long as that individual knows the correct password. Shares often have multiple passwords. For example, one password may grant read-only access, while another may grant read-write access, and so on. Security is maintained as long as unauthorized users do not discover the password for a share to which they shouldn't have access.

OS/2 and Window 95/98 both support share-level security on their resources. You can set up share-level security with Windows 95/98 by first enabling share-level security using the Access Control tab of the Network Control Panel dialog. Then select the Share-level Access Control radio button (which deselects the user-level access control radio button), as shown in Figure 6-1, and press the OK button.



Control access to shared resources using:

[✓ Share-level access control]
Enables you to supply a password for each shared resource.

User-level access control
Enables you to specify users and groups who have access to each shared resource.

Obtain list of users and groups from:

Figure 6-1. Selecting share-level security on a Windows machine

Next, right click on a resource—such as a hard drive or a CD-ROM—and select the Properties menu item. This will bring up the Resource Properties dialog box. Select the Sharing tab at the top of the dialog box and enable the resource as Shared As. From here, you can configure how the shared resource will appear to individual users, as well as assigning whether the resource will appear as read-only, read-write, or a mix, depending on the password that is supplied.

You might be thinking that this security model is not a good fit for Samba—and you would be right. In fact, if you set the `security = share` option in the Samba configuration file, Samba will still reuse the username/passwords combinations in the system password files to authenticate access. More precisely, Samba will take the following steps when a client requests a connection using share-level security:

1. When a connection is requested, Samba will accept the password and (if sent) the username of the client.
2. If the share is `guest only`, the user is immediately granted access to the share with the rights of the user specified by the `guest account` parameter; no password checking is performed.
3. For other shares, Samba appends the username to a list of users who are allowed access to the share. It then attempts to validate the password given in association with that username. If successful, Samba grants the user access to the share with the rights assigned to that user. The user will not need to authenticate again unless a `revalidate = yes` option has been set inside the share.
4. If the authentication is unsuccessful, Samba will attempt to validate the password against the list of users it has previously compiled throughout the attempted connections, as well as any specified under the share in the configuration file. If the password does not match any usernames (as specified in the system password file, typically */etc/passwd*), the user is not granted access to the share under that username.
5. However, if the share has a `guest ok` or `public` option set, the user will default to access with the rights of the user specified by the `guest account` option.

You can indicate in the configuration file which users should be initially placed on the share-level security user list by using the `username` configuration option, as shown below:

```
[global]
    security = share
[accounting1]
    path = /home/samba/accounting1
    guest ok = no
    writable = yes
    username = davecb, pkelly, andyo
```

Here, when a user attempts to connect to a share, Samba will verify the password that was sent against each of the users in its own list, in addition to the passwords of users `davecb`, `pkelly`, and `andyo`. If any of the passwords match, the connection will be verified and the user will be allowed. Otherwise, connection to the specific share will fail.

Share Level Security Options

Table 6-4 shows the options typically associated with share-level security.

Table 6-4. Share-Level Access Options

Option	Parameters	Function	Default	Scope
`only user`	boolean	Indicates whether usernames specified by `username` will be the only ones allowed.	`no`	Share
`username` (`user` or `users`)	string (list of usernames)	Specifies a list of users against which a client's password will be tested.	None	Share

only user

This boolean option indicates whether Samba will allow connections to a share using share-level security based solely on the individuals specified in the `username` option, instead of those users compiled on Samba's internal list. The default value for this option is `no`. You can override it per share as follows:

```
[global]
    security = share
[data]
    username = andy, peter, valerie
    only user = yes
```

username

This option presents a list of users against which Samba will test a connection password to allow access. It is typically used with clients that have share-level security to allow connections to a particular service based solely on a qualifying password—in this case, one that matches a password set up for a specific user:

```
[global]
    security = share
[data]
     username = andy, peter, terry
```

We recommend against using this option unless you are implementing a Samba server with share-level security.

User-level Security

The preferred mode of security with Samba is *user-level security*. With this method, each share is assigned specific users that can access it. When a user requests a connection to a share, Samba authenticates by validating the given username and password with the authorized users in the configuration file and the passwords in the password database of the Samba server. As mentioned earlier in the chapter, one way to isolate which users are allowed access to a specific share is by using the `valid users` option for each share:

```
[global]
    security = user
[accounting1]
    writable = yes
    valid users = bob, joe, sandy
```

Each of the users listed will be allowed to connect to the share if the password provided matches the password stored in the system password database on the server. Once the initial authentication succeeds, the user will not need to re-enter a password again to access that share unless the `revalidate = yes` option has been set.

Passwords can be sent to the Samba server in either an encrypted or a non-encrypted format. If you have both types of systems on your network, you should ensure that the passwords represented by each user are stored both in a traditional account database and Samba's encrypted password database. This way, authorized users can gain access to their shares from any type of client.* However, we recommend that you move your system to encrypted passwords and abandon non-encrypted passwords if security is an issue. The "Passwords" section of this chapter explains how to use encrypted as well as non-encrypted passwords.

Server-level Security

Server-level security is similar to user-level security. However, with server-level security, Samba delegates password authentication to another SMB password server, typically another Samba server or a Windows NT Server acting as a PDC on the network. Note that Samba still maintains its list of shares and their configuration in its *smb.conf* file. When a client attempts to make a connection to a particular share, Samba validates that the user is indeed authorized to connect to the share. Samba will then attempt to validate the password by contacting the SMB password server through a known protocol and presenting the username and password to the SMB password server. If the password is accepted, a session will be established with the client. See Figure 6-2 for an illustration of this setup.

You can configure Samba to use a separate password server under server-level security with the use of the `password server` global configuration option, as follows:

* Having both encrypted and non-encrypted password clients on your network is another reason why Samba allows you to include (or not include) various options in the Samba configuration file based on the client operating system or machine name variables.

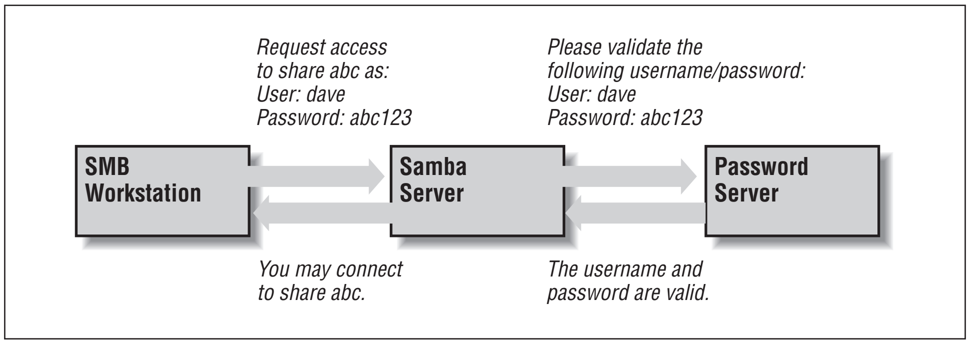


Figure 6-2. A typical system setup using server level security

```
[global]
    security = server
    password server = PHOENIX120 HYDRA134
```

Note that you can specify more than one machine as the target of the `password server`; Samba will move down the list of servers in the event that its first choice is unreachable. The servers identified by the `password server` option are given as NetBIOS names, not their DNS names or equivalent IP addresses. Also, if any of the servers reject the given password, the connection will automatically fail—Samba will not attempt another server.

One caveat: when using this option, you will still need an account representing that user on the regular Samba server. This is because the Unix operating system needs a username to perform various I/O operations. The preferable method of handling this is to give the user an account on the Samba server but disable the account's password by replacing it in the system password file (e.g., */etc/passwd*) with an asterisk (*).

Domain-level Security

Domain-level security is similar to server-level security. However, with domain-level security, the Samba server is acting as a member of a Windows domain. Recall from Chapter 1 that each domain has a *domain controller*, which is usually a Windows NT server offering password authentication. Including these controllers provides the workgroup with a definitive password server. The domain controllers keep track of users and passwords in their own security authentication module (SAM), and authenticates each user when he or she first logs on and wishes to access another machine's shares.

As mentioned earlier in this chapter, Samba has a similar ability to offer user-level security, but this option is Unix-centric and assumes that the authentication occurs via Unix password files. If the Unix machine is part of a NIS or NIS+ domain, Samba will authenticate the users transparently against a shared password file, in typical Unix fashion. Samba then provides access to the NIS or NIS+ domain from Windows. There is, of course, no relationship between the NIS concept of a domain and the Windows concept of a domain.

With domain-level security, we now have the option of using the native NT mechanism. This has a number of advantages:

- It provides far better integration with NT: there are fewer "kludges" in the *smb.conf* options dealing with domains than with most Windows features. This allows more extensive use of NT management tools, such as the User Manager for Domains tool allowing PC support individuals to treat Samba servers as if they were large NT machines.
- With the better integration comes protocol and code cleanups, allowing the Samba team to track the evolving NT implementation. NT Service Pack 4 corrects several problems in the protocol, and Samba's better integration makes it easier to track and adapt to these changes.
- There is less overhead on the PDC because there is one less permanent network connection between it and the Samba server. Unlike the protocol used by the `security = server` option, the Samba server can make a Remote Procedure Call (RPC) call only when it needs authentication information. It can not keep a connection permanently up just for that.
- Finally, the NT domain authentication scheme returns the full set of user attributes, not just success or failure. The attributes include a longer, more network-oriented version of the Unix uid, NT groups, and other information. This includes:
  - Username
  - Full name
  - Description
  - Security identifier (a domain-wide extension of the Unix uid)
  - NT group memberships
  - Logon hours, and whether to force the user to log out immediately
  - Workstations the user is allowed to use
  - Account expiration date
  - Home directory
  - Login script
  - Profile
  - Account type

- The Samba developers used domain-level security in Samba version 2.0.4 to add and delete domain users on Samba servers semi-automatically. In addition, it adds room for other NT-like additions, such as supporting access control lists and changing permissions of files from the client.

The advantage to this approach is less administration; there is only one authentication database to keep synchronized. The only local administration required on the Samba server will be creating directories for users to work in and */etc/passwd* entries to keep their UIDs and groups in.

Adding a Samba server to a Windows NT Domain

If you already have an NT domain, you can easily add a Samba server to it. First, you will need to stop the Samba daemons. Then, add the Samba server to the NT domain on the PDC using the "Windows NT Server Manager for Domains" tool. When it asks for the computer type, choose "Windows NT Workstation or Server," and give it the NetBIOS name of the Samba server. This creates the machine account on the NT server.

Next, generate a Microsoft-format machine password using the *smbpasswd* tool, which is explained in further detail in the next section. For example, if our domain is SIMPLE and the Windows NT PDC is `beowulf`, we could use the following command on the Samba server to accomplish this:

```
smbpasswd -j SIMPLE -r beowulf
```

Finally, add the following options to the `[global]` section of your *smb.conf* and restart the Samba daemons.

```
[global]
    security = domain
    domain logins = yes
    workgroup = SIMPLE
    password server = beowulf
```

Samba should now be configured for domain-level security. The `domain logins` option is explained in more detail later in this chapter.

Passwords

Passwords are a thorny issue with Samba. So much so, in fact, that they are almost always the first major problem that users encounter when they install Samba, and generate by far the most questions sent to Samba support groups. In previous chapters, we've gotten around the need for passwords by placing the `guest ok` option in each of our configuration files, which allows connections without authenticating passwords. However, at this point, we need to delve deeper into Samba to discover what is happening on the network.

Passwords sent from individual clients can be either encrypted or non-encrypted. Encrypted passwords are, of course, more secure. A non-encrypted password can be easily read with a packet sniffing program, such as the modified *tcpdump* program for Samba that we used in Chapter 3, *Configuring Windows Clients*. Whether passwords are encrypted depends on the operating system that the client is using to connect to the Samba server. Table 6-5 lists which Windows operating systems encrypt their passwords before sending them to the primary domain controller for authentication. If your client is not Windows, check the system documentation to see if SMB passwords are encrypted.

Table 6-5. Windows Operating Systems with Encrypted Passwords

Operating System	Encrypted or Non-encrypted
Windows 95	Non-encrypted
Windows 95 with SMB Update	Encrypted
Windows 98	Encrypted
Windows NT 3.*x*	Non-encrypted
Windows NT 4.0 before SP 3	Non-encrypted
Windows NT 4.0 after SP 3	Encrypted

There are actually two different encryption methods used: one for Windows 95 and 98 clients that reuses Microsoft's LAN Manager encryption style, and a separate one for Windows NT clients and servers. Windows 95 and 98 use an older encryption system inherited from the LAN Manager network software, while Windows NT clients and servers use a newer encryption system.

If encrypted passwords are supported, Samba stores the encrypted passwords in a file called *smbpasswd*. By default, this file is located in the *private* directory of the Samba distribution (*/usr/local/samba/private*). At the same time, the client stores an encrypted version of a user's password on its own system. The plaintext password is never stored on either system. Each system encrypts the password automatically using a known algorithm when the password is set or changed.

When a client requests a connection to an SMB server that supports encrypted passwords (such as Samba or Windows NT), the two computers undergo the following negotiations:

1. The client attempts to negotiate a protocol with the server.
2. The server responds with a protocol and indicates that it supports encrypted passwords. At this time, it sends back a randomly-generated 8-byte challenge string.

3. The client uses the challenge string as a key to encrypt its already encrypted password using an algorithm predefined by the negotiated protocol. It then sends the result to the server.
4. The server does the same thing with the encrypted password stored in its database. If the results match, the passwords are equivalent and the user is authenticated.

Note that even though the original passwords are not involved in the authentication process, you need to be very careful that the encrypted passwords located inside of the *smbpasswd* file are guarded from unauthorized users. If they are compromised, an unauthorized user can break into the system by replaying the steps of the previous algorithm. The encrypted passwords are just as sensitive as the plaintext passwords—this is known as *plaintext-equivalent* data in the cryptography world. Of course, you should also ensure that the clients safeguard their plaintext-equivalent passwords as well.

You can configure Samba to accept encrypted passwords with the following global additions to *smb.conf*. Note that we explicitly name the location of the Samba password file:

```
[global]
    security = user
    encrypt passwords = yes
    smb passwd file = /usr/local/samba/private/smbpasswd
```

Samba, however, will not accept any users until the *smbpasswd* file has been initialized.

Disabling encrypted passwords on the client

While Unix authentication has been in use for decades, including the use of *telnet* and *rlogin* access across the Internet, it embodies well-known security risks. Plaintext passwords are sent over the Internet and can be retrieved from TCP packets by malicious snoopers. However, if you feel that your network is secure and you wish to use standard Unix */etc/passwd* authentication for all clients, you can do so, but you must disable encrypted passwords on those Windows clients that default to using them.

In order to do this, you must modify the Windows registry by installing two files on each system. Depending on the platform involved, the files are either *NT4_PlainPassword.reg* or *Win95_PlainPassword.reg*. You can perform this installation by copying the appropriate *.reg* files from the Samba distribution's */docs* directory to a DOS floppy, and running it from the Run menu item on the client's Start Menu button. Incidentally, the Windows 95 *.reg* file works fine on Windows 98 as well.

After you reboot the machine, the client will not encrypt its hashed passwords before sending them to the server. This means that the plaintext-equivalent passwords can been seen in the TCP packets that are broadcast across the network. Again, we encourage you not to do this unless you are absolutely sure that your network is secure.

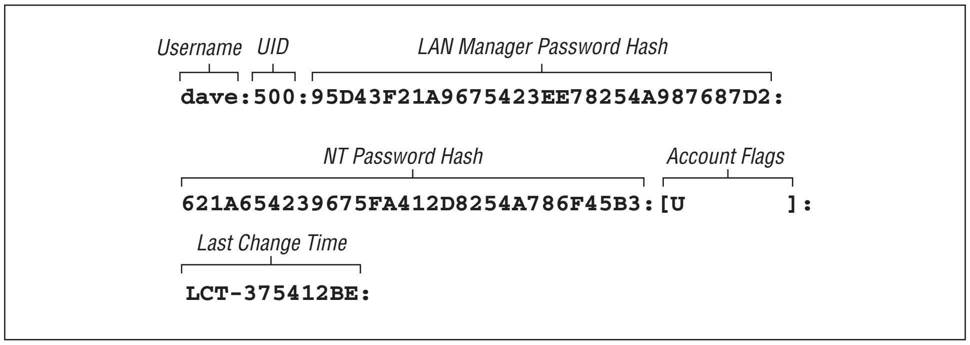
If passwords are not encrypted, you can indicate as much in your Samba configuration file:

```
[global]
    security = user
    encrypt passwords = no
```

The smbpasswd File

Samba stores its encrypted passwords in a file called *smbpasswd*, which by default resides in the */usr/local/samba/private* directory. The *smbpasswd* file should be guarded as closely as the *passwd* file; it should be placed in a directory to which only the root user has read/write access. All other users should not be able to read from the directory at all. In addition, the file should have all access closed off to all users except for root.

Before you can use encrypted passwords, you will need to create an entry for each Unix user in the *smbpasswd* file. The structure of the file is somewhat similar to a Unix *passwd* file, but has different fields. Figure 6-3 illustrates the layout of the *smbpasswd* file; the entry shown is actually one line in the file.



```
dave:500:95D43F21A9675423EE78254A987687D2:
621A654239675FA412D8254A786F45B3:[U         ]:
LCT-375412BE:
```

Figure 6-3. Structure of the smbpasswd file entry (actually one line)

Here is a breakdown of the individual fields:

Username
This is the username of the account. It is taken directly from the system password file.

UID

This is the user ID of the account. Like the username, it is taken directly from the system password file and must match the user it represents there.

LAN Manager Password Hash

This is a 32-bit hexadecimal sequence that represents the password Windows 95 and 98 clients will use. It is derived by encrypting the string `KGS!@#$%` with a 56-bit DES algorithm using the user's password (forced to 14 bytes and converted to capital letters) twice repeated as the key. If there is currently no password for this user, the first 11 characters of the hash will consist of the sequence `NO PASSWORD` followed by `X` characters for the remainder. Anyone can access the share with no password. On the other hand, if the password has been disabled, it will consist of 32 `X` characters. Samba will not grant access to a user without a password unless the `null passwords` option has been set.

NT Password Hash

This is a 32-bit hexadecimal sequence that represents the password Windows NT clients will use. It is derived by hashing the user's password (represented as a 16-bit little-endian Unicode sequence) with an MD4 hash. The password is not converted to uppercase letters first.

Account Flags

This field consists of 11 characters between two braces ([ ]). Any of the following characters can appear in any order; the remaining characters should be spaces:

U This account is a standard user account.

D This account is currently disabled and Samba should not allow any logins.

N This account has no password associated with it.

W This is a workstation trust account that can be used to configure Samba as a primary domain controller (PDC) when allowing Windows NT machines to join its domain.

Last Change Time

This code consists of the characters `LCT-` followed by a hexidecimal representation of the amount of seconds since the epoch (midnight on January 1, 1970) that the entry was last changed.

Adding entries to smbpasswd

There are a few ways you can add a new entry to the *smbpasswd* file:

- You can use the *smbpasswd* program with the `-a` option to automatically add any user that currently has a standard Unix system account on the server. This program resides in the */usr/local/samba/bin* directory.

- You can use the *addtosmbpass* executable inside the */usr/local/samba/bin* directory. This is actually a simple *awk* script that parses a system password file and extracts the username and UID of each entry you wish to add to the SMB password file. It then adds default fields for the remainder of the user's entry, which can be updated using the *smbpasswd* program later. In order to use this program, you will probably need to edit the first line of the file to correctly point to *awk* on your system.
- In the event that the neither of those options work for you, you can create a default entry by hand in the *smbpasswd* file. The entry should be entirely on one line. Each field should be colon-separated and should look similar to the following:

  ```
  dave:500:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX:[U
  ]:LCT-00000000:
  ```

  This consists of the username and the UID as specified in the system password file, followed by two sets of exactly 32 `X` characters, followed by the account flags and last change time as it appears above. After you've added this entry, you must use the *smbpasswd* program to change the password for the user.

Changing the encrypted password

If you need to change the encrypted password in the *smbpasswd* file, you can also use the *smbpasswd* program. Note that this program shares the same name as the encrypted password file itself, so be sure not to accidentally confuse the password file with the password-changing program.

The *smbpasswd* program is almost identical to the *passwd* program that is used to change Unix account passwords. The program simply asks you to enter your old password (unless you're the root user), and duplicate entries of your new password. No password characters are shown on the screen.

```
# smbpasswd dave
Old SMB password:
New SMB password:
Retype new SMB password:
Password changed for user dave
```

You can look at the *smbpasswd* file after this command completes to verify that both the LAN Manager and the NT hashes of the passwords have been stored in their respective positions. Once users have encrypted password entries in the database, they should be able to connect to shares using encrypted passwords!

Password Synchronization

Having a regular password and an encrypted version of the same password can be troublesome when you need to change both of them. Luckily, Samba affords you a limited ability to keep your passwords synchronized. Samba has a pair of configuration options that can be used to automatically update a user's regular Unix password when the encrypted password is changed on the system. The feature can be activated by specifying the `unix password sync` global configuration option:

```
[global]
    encrypt passwords = yes
    smb passwd file = /usr/local/samba/private/smbpasswd

    unix password sync = yes
```

With this option enabled, Samba will attempt to change the user's regular password (as `root`) when the encrypted version is changed with *smbpasswd*. However, there are two other options that have to be set correctly in order for this to work.

The easier of the two is `passwd program`. This option simply specifies the Unix command used to change a user's standard system password. It is set to `/bin/passwd %u` by default. With some Unix systems, this is sufficient and you do not need to change anything. Others, such as Red Hat Linux, use */usr/bin/passwd* instead. In addition, you may want to change this to another program or script at some point in the future. For example, let's assume that you want to use a script called `changepass` to change a user's password. Recall that you can use the variable `%u` to represent the current Unix username. So the example becomes:

```
[global]
    encrypt passwords = yes
    smb passwd file = /usr/local/samba/private/smbpasswd

    unix password sync = yes
    passwd program = changepass %u
```

Note that this program will be called as the `root` user when the `unix password sync` option is set to `yes`. This is because Samba does not necessarily have the plaintext old password of the user.

The harder option to configure is `passwd chat`. The `passwd chat` option works like a Unix chat script. It specifies a series of strings to send as well as responses to expect from the program specified by the `passwd program` option. For example, this is what the default `passwd chat` looks like. The delimiters are the spaces between each groupings of characters:

```
passwd chat = *old*password* %o\n *new*password* %n\n *new*password* %n\n
*changed*
```

The first grouping represents a response expected from the password-changing program. Note that it can contain wildcards (*), which help to generalize the chat programs to be able to handle a variety of similar outputs. Here, `*old*password*` indicates that Samba is expecting any line from the password program containing the letters `old` followed by the letters `password`, without regard for what comes on either side or between them. Once instructed to, Samba will wait indefinitely for such a match. Is Samba does not receive the expected response, the password will fail.

The second grouping indicates what Samba should send back once the data in the first grouping has been matched. In this case, you see `%o\n`. This response is actually two items: the variable `%o` represents the old password, while the `\n` is a newline character. So, in effect, this will "type" the old password into the standard input of the password changing program, and then "press" Enter.

Following that is another response grouping, followed by data that will be sent back to the password changing program. (In fact, this response/send pattern continues indefinitely in any standard Unix *chat* script.) The script continues until the final pattern is matched.*

You can help match the response strings sent from the password program with the characters listed in Table 6-6. In addition, you can use the characters listed in Table 6-7 to help formulate your response.

Table 6-6. Password Chat Response Characters

Character	Definition
`*`	Zero or more occurrences of any character.
`" "`	Allows you to include matching strings that contain spaces. Asterisks are still considered wildcards even inside of quotes, and you can represent a null response with empty quotes.

Table 6-7. Password Chat Send Characters

Character	Definition
`%o`	The user's old password
`%n`	The user's new password
`\n`	The linefeed character
`\r`	The carriage-return character
`\t`	The tab character
`\s`	A space

For example, you may want to change your password chat to the following entry. This will handle scenarios in which you do not have to enter the old password. In

* This may not work under Red Hat Linux, as the password program typically responds "All authentication tokens updated successfully," instead of "Password changed." We provide a fix for this later in this section.

addition, this will also handle the new `all tokens updated successfully` string that Red Hat Linux sends:

```
passwd chat = *new password* %n\n *new password* %n\n *success*
```

Again, the default chat should be sufficient for many Unix systems. If it isn't, you can use the `passwd chat debug` global option to set up a new chat script for the password change program. The `passwd chat debug` option logs everything during a password chat. This option is a simple boolean, as shown below:

```
[global]
    encrypted passwords = yes
    smb passwd file = /usr/local/samba/private/smbpasswd

    unix password sync = yes
    passwd chat debug = yes
    log level = 100
```

After you activate the password chat debug feature, all I/O received by Samba through the password chat will be sent to the Samba logs with a debug level of 100, which is why we entered a new log level option as well. As this can often generate multitudes of error logs, it may be more efficient to use your own script, by setting the `passwd program` option, in place of */bin/passwd* to record what happens during the exchange. Also, make sure to protect your log files with strict file permissions and to delete them as soon as you've grabbed the information you need, because they contain the passwords in plaintext.

The operating system on which Samba is running may have strict requirements for valid passwords in order to make them more impervious to dictionary attacks and the like. Users should be made aware of these restrictions when changing their passwords.

Earlier we said that password synchronization is limited. This is because there is no reverse synchronization of the encrypted *smbpasswd* file when a standard Unix password is updated by a user. There are various strategies to get around this, including NIS and freely available implementations of the pluggable authentication modules (PAM) standard, but none of them really solve all the problems yet. In the future, when Windows 2000 emerges, we will see more compliance with the Lightweight Directory Access Protocol (LDAP), which promises to make password synchronization a thing of the past.

Password Configuration Options

The options in Table 6-8 will help you work with passwords in Samba.

Table 6-8. Password Configuration Options

Option	Parameters	Function	Default	Scope
`encrypt passwords`	boolean	Turns on encrypted passwords.	`no`	Global
`unix password sync`	boolean	If `yes`, Samba updates the standard Unix password database when a user changes his or her encrypted password.	`no`	Global
`passwd chat`	string (chat commands)	Sets a sequence of commands that will be sent to the password program.	See earlier section on this option	Global
`passwd chat debug`	boolean	Sends debug logs of the password-change process to the log files with a level of 100.	`no`	Global
`passwd program`	string (Unix command)	Sets the program to be used to change passwords.	`/bin/passwd %u`	Global
`password level`	numeric	Sets the number of capital letter permutations to attempt when matching a client's password.	None	Global
`update encrypted`	boolean	If `yes`, Samba updates the encrypted password file when a client connects to a share with a plaintext password.	`no`	Global
`null passwords`	boolean	If `yes`, Samba allows access for users with null passwords.	`no`	Global
`smb passwd file`	string (fully-qualified pathname)	Specifies the name of the encrypted password file.	`/usr/local/samba/private/smbpasswd`	Global
`hosts equiv`	string (fully-qualified pathname)	Specifies the name of a file that contains hosts and users that can connect without using a password.	None	Global
`use rhosts`	string (fully-qualified pathname)	Specifies the name of an *.rhosts* file that allows users to connect without using a password.	None	Global

unix password sync

The `unix password sync` global option allows Samba to update the standard Unix password file when a user changes his or her encrypted password. The encrypted password is stored on a Samba server in the *smbpasswd* file, which is located in */usr/local/samba/private* by default. You can activate this feature as follows:

```
[global]
    unix password sync = yes
```

If this option is enabled, Samba changes the encrypted password and, in addition, attempts to change the standard Unix password by passing the username and new password to the program specified by the `passwd program` option (described earlier). Note that Samba does not necessarily have access to the plaintext password for this user, so the password changing program must be invoked as `root`.* If the Unix password change does not succeed, for whatever reason, the SMB password will not be changed either.

encrypt passwords

The `encrypt passwords` global option switches Samba from using plaintext passwords to encrypted passwords for authentication. Encrypted passwords will be expected from clients if the option is set to `yes`:

```
encrypt passwords = yes
```

By default, Windows NT 4.0 with Service Pack 3 or above and Windows 98 transmit encrypted passwords over the network. If you are enabling encrypted passwords, you must have a valid *smbpasswd* file in place and populated with usernames that will authenticate with encrypted passwords. (See the section "The smbpasswd File," earlier in this chapter.) In addition, Samba must know the location of the *smbpasswd* file; if it is not in the default location (typically */usr/local/samba/private/smbpasswd*), you can explicitly name it using the `smb passwd file` option.

If you wish, you can use the `update encrypted` to force Samba to update the *smbpasswd* file with encrypted passwords each time a client connects to a non-encrypted password.

A common strategy to ensure that hosts who need encrypted password authentication indeed receive it is with the `include` option. With this, you can create individual configuration files that will be read in based on OS-type (`%a`) or client name (`%m`). These host-specific or OS-specific configuration files can contain an `encrypted passwords = yes` option that will activate only when those clients are connecting to the server.

* This is because the Unix *passwd* program, which is the usual target for this operation, allows `root` to change a user's password without the security restriction that requests the old password of that user.

passwd program

The `passwd program` is used to specify a program on the Unix Samba server that Samba can use to update the standard system password file when the encrypted password file is updated. This option defaults to the standard *passwd* program, usually located in the */bin* directory. The `%u` variable is typically used here as the requesting user when the command is executed. The actual handling of input and output to this program during execution is handled through the `passwd chat` option. The "Password Synchronization" section, earlier in this chapter, covers this option in detail.

passwd chat

This option specifies a series of send/response strings similar to a Unix chat script, which are used to interface with the password-changing program on the Samba server. The "Password Synchronization" section, earlier in this chapter, covers this option in detail.

passwd chat debug

If set to `yes`, the `passwd chat debug` global option logs everything sent or received by Samba during a password chat. All the I/O received by Samba through the password chat is sent to the Samba logs with a debug level of 100; you will need to specify `log level = 100` in order for the information to be recorded. The "Password Synchronization" section, earlier in this chapter, describes this option in more detail. Be aware that if you do set this option, the plaintext passwords will be visible in the debugging logs, which could be a security hazard if they are not properly secured.

password level

With SMB, non-encrypted (or plaintext) passwords are sent with capital letters, just like the usernames mentioned previously. Many Unix users, however, choose passwords with both uppercase and lowercase letters. Samba, by default, only attempts to match the password entirely in lowercase letters, and not capitalizing the first letter.

Like `username level`, there is a `password level` option that can be used to attempt various permutations of the password with capital letters. This option takes an integer value that specifies how many letters in the password should be capitalized when attempting to connect to a share. You can specify this options as follows:

```
[global]
    password level = 3
```

In this case, Samba will then attempt all permutations of the password it can compute having three capital letters. The larger the number, the more computations Samba will have to perform to match the password, and the longer a connection to a specific share may take.

update encrypted

For sites switching over to the encrypted password format, Samba provides an option that should help with the transition. The `update encrypted` option allows a site to ease into using encrypted passwords from plaintext passwords. You can activate this option as follows:

```
[global]
    update encrypted = yes
```

This instructs Samba to create an encrypted version of each user's Unix password in the *smbpasswd* file each time he or she connects to a share. When this option is enabled, you must have the `encrypt passwords` option set to `no` so that the client will pass plaintext passwords to Samba to use to update the files. Once each user has connected at least once, you can set `encrypted passwords = yes`, allowing you to use only the encrypted passwords. The user must already have a valid entry in the *smbpasswd* file for this option to work.

null passwords

This global option tells Samba whether or not to allow access from users that have null passwords (encrypted or non-encrypted) set in their accounts. The default value is `no`. You can override it as follows:

```
null passwords = yes
```

We highly recommend against doing so unless you are familiar with the security risks this option can present to your system, including inadvertent access to system users (such as *bin*) in the system password file who have null passwords set.

smb passwd file

This global option identifies the location of the encrypted password database. By default, it is set to */usr/local/samba/private/smbpasswd.* You can override it as follows:

```
[global]
    smb passwd file = /etc/smbpasswd
```

This location, for example, is common on many Red Hat distributions.

hosts equiv

This global option specifies the name of a standard Unix *hosts.equiv* file that will allow hosts or users to access shares without specifying a password. You can specify the location of such a file as follows:

```
[global]
    hosts equiv = /etc/hosts.equiv
```

The default value for this option does not specify any *hosts.equiv* file. Because using such a file is essentially a huge security risk, we highly recommend that you do not use this option unless you are confident in the security of your network.

use rhosts

This global option specifies the name of a standard Unix user's *.rhosts* file that will allow foreign hosts to access shares without specifying a password. You can specify the location of such a file as follows:

```
[global]
    use rhosts = /home/dave/.rhosts
```

The default value for this option does not specify any *.rhosts* file. Like the `hosts equiv` option above, using such a file is a security risk. We highly recommend that you do use this option unless you are confident in the security of your network.

Windows Domains

Now that you are comfortable with users and passwords on a Samba server, we can show you how to set up Samba to become a primary domain controller for Windows 95/98 and NT machines. Why use domains? The answer probably isn't obvious until you look behind the scenes, especially with Windows 95/98.

Recall that with traditional workgroups, Windows 95/98 simply accepts each username and password that you enter when logging on to the system. There are no unauthorized users with Windows 95/98; if a new user logs on, the operating system simply asks for a new password and authenticates the user against that password from then on. The only time that Windows 95/98 attempts to use the password you entered is when connecting to another share.

Domain logons, on the other hand, are similar to Unix systems. In order to log on to the domain, a valid username and password must be presented at startup, which is then authenticated against the primary domain controller's password database. If the password is invalid, the user is immediately notified and they cannot log on to the domain.

There's more good news: once you have successfully logged on to the domain, you can access any of the shares in the domain to which you have rights without having to reauthenticate yourself. More precisely, the primary domain controller returns a token to the client machine that allows it to access any share without consulting the PDC again. Although you probably won't notice the shift, this can be beneficial in cutting down network traffic. (You can disable this behavior if you wish by using the `revalidate` option.)

Configuring Samba for Windows Domain Logons

If you wish to allow Samba to act as a domain controller, use the following sections to configure Samba and your clients to allow domain access.

If you would like more information on how to set up domains, see the *DOMAINS.TXT* file that comes with the Samba distribution.

Windows 95/98 clients

Setting up Samba as a PDC for Windows 95/98 clients is somewhat anticlimactic. All you really need to do on the server side is ensure that:

- Samba is the only primary domain controller for the current workgroup.
- There is a WINS server available on the network, either a Samba machine or a Windows NT server. (See Chapter 7, *Printing and Name Resolution*, for more information on WINS.)
- Samba is using user-level security (i.e., it doesn't hand off password authentication to anyone else). You do not want to use domain-level security if Samba itself is acting as the PDC.

At that point, you can insert the following options into your Samba configuration file:

```
[global]
    workgroup = SIMPLE
    domain logons = yes

# Be sure to set user-level security!

    security = user

# Be sure to become the primary domain controller!

    os level = 34
    local master = yes
    preferred master = yes
    domain master = yes
```

The `domain logons` option enables Samba to perform domain authentication on behalf of other clients that request it. The name of the domain will be the same as the workgroup listed in the Samba configuration file, in this case: SIMPLE.

After that, you need to create a non-writable, non-public, non-browesable disk share called `[netlogon]` (it does not matter where this share points to as long as each Windows client can connect to it):

```
[netlogon]
    comment = The domain logon service
    path = /export/samba/logon
    public = no
    writeable = no
    browsable = no
```

Windows NT clients

If you have Window NT clients on your system, there are a few more steps that need to be taken in order for Samba to act as their primary domain controller.

> You will need to use at least Samba 2.1 to ensure that PDC functionality for Windows NT clients is present. Prior to Samba 2.1, only limited user authentication for NT clients was present. At the time this book went to press, Samba 2.0.5 was the latest version, but Samba 2.1 was available through CVS download. Instructions on downloading alpha versions of Samba are given in Appendix E, *Downloading Samba with CVS*.

As before, you need to ensure that Samba is a primary domain controller for the current workgroup and is using user-level security. However, you must also ensure that Samba is using encrypted passwords. In other words, alter the `[global]` options the previous example to include the `encrypted passwords = yes` option, as shown here:

```
[global]
    workgroup = SIMPLE
    encrypted passwords = yes
    domain logons = yes

    security = user
```

Creating trust accounts for NT clients

This step is exclusively for Windows NT clients. All NT clients that connect to a primary domain controller make use of *trust accounts*. These accounts allow a machine to log in to the PDC itself (not one of its shares), which means that the PDC can trust any further connections from users on that client. For all intents and purposes, a trust account is identical to a user account. In fact, we will be using standard Unix user accounts to emulate trust accounts for the Samba server.

The login name of a machine's trust account is the name of the machine with a dollar sign appended to it. For example, if our Windows NT machine is named `chimaera`, the login account would be `chimaera$`. The initial password of the account is simply the name of the machine in lowercase letters. In order to forge the trust account on the Samba server, you need to create a Unix account with the appropriate machine name, as well as an encrypted password entry in the *smbpasswd* database.

Let's tackle the first part. Here, we only need to modify the */etc/passwd* file to support the trust account; there is no need to create a home directory or assign a shell to the "user" because the only part we are interested in is whether a login is permitted. Therefore, we can create a "dummy" account with the following entry:

```
chimaera$:*:1000:900:Trust Account:/dev/null:/dev/null
```

Note that we have also disabled the password field by placing a `*` in it. This is because Samba will use the *smbpasswd* file to contain the password instead, and we don't want anyone to telnet into the machine using that account. In fact, the only value other than the account name that is used here is the UID of the account for the encrypted password database (1000). This number must map to a unique resource ID on the NT server and cannot conflict with any other resource IDs. Hence, no NT user or group should map to this number or a networking error will occur.

Next, add the encrypted password using the *smbpasswd* command, as follows:

```
# smbpasswd -a -m chimaera
Added user chimaera$
Password changed for user chimaera$
```

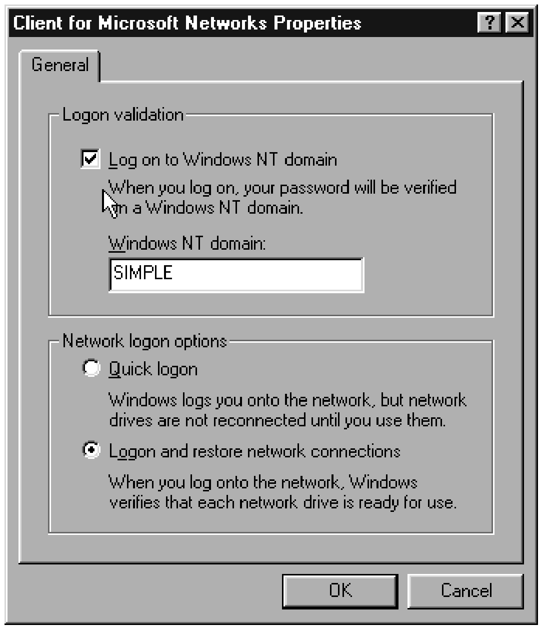
The `-m` option specifies that a machine trust account is being generated. The *smbpasswd* program will automatically set the initial encrypted password as the NetBIOS name of the machine in lowercase letters; you don't need to enter it. When specifying this option on the command line, do not put a dollar sign after the machine name—it will be appended automatically. Once the encrypted password has been added, Samba is ready to handle domain logins from a NT client.

Configuring Windows Clients for Domain Logons

Once you have Samba configured for domain logons, you need to set up your Windows clients to log on to the domain at startup.

Windows 95/98

With Windows 95/98, this can be done by raising the Network configuration dialog in the Windows Control Panel and selecting the Properties for "Client for Microsoft Networks." At this point, you should see a dialog box similar to Figure 6-4. Select the "Logon to Windows Domain" checkbox at the top of the dialog box, and enter the workgroup that is listed in the Samba configuration file as the Windows NT domain. Then click on OK and reboot the machine when asked.



☑ Log on to Windows NT domain

When you log on, your password will be verified on a Windows NT domain.

Windows NT domain: SIMPLE

Network logon options

○ Quick logon

Windows logs you onto the network, but network drives are not reconnected until you use them.

● Logon and restore network connections

When you log onto the network, Windows verifies that each network drive is ready for use.

Figure 6-4. Configuring a Windows 95/98 client for domain logons

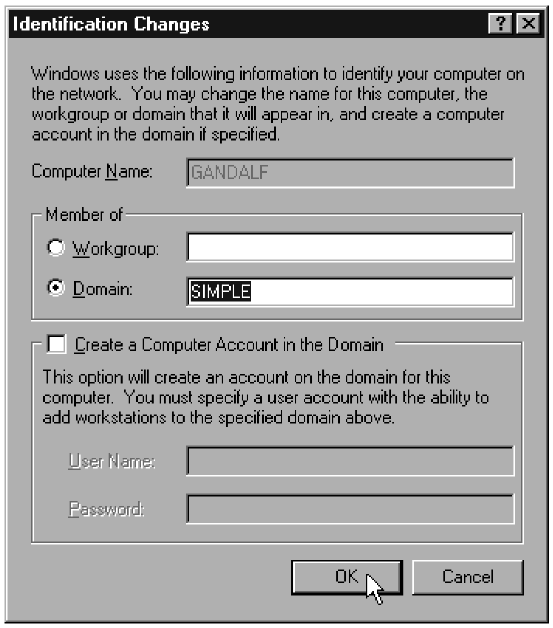
If Windows complains that you are already logged into the domain, you probably have an active connection to a share in the workgroup (such as a mapped network drive). Simply disconnect the resource temporarily by right-clicking on its icon and choosing the Disconnect pop-up menu item.

When Windows reboots, you should see the standard login dialog with an addition: a field for a domain. The domain name should already be filled in, so simply enter your password and click on the OK button. At this point, Windows should consult the primary domain controller (Samba) to see if the password is correct. (You can check the log files if you want to see this in action.) If it worked, congratulations! You have properly configured Samba to act as a domain controller for Windows 95/98 machines and your client is successfully connected.

Windows NT 4.0

To configure Windows NT for domain logons, open the Network configuration dialog in the Windows NT Control Panel. The first tab that you see should list the identification of the machine.

Press the Change button and you should see the dialog box shown in Figure 6-5. In this dialog box, you can choose to have the Windows NT client become a member of the domain by selecting the radio button marked Domain in the "Member of" box. Then, type in the domain that you wish the client to login to; it should be the same as the workgroup that you specified in the Samba configuration file. Do not check the box marked "Create a Computer Account in the Domain"—Samba does not currently support this functionality.



Identification Changes

Windows uses the following information to identify your computer on the network. You may change the name for this computer, the workgroup or domain that it will appear in, and create a computer account in the domain if specified.

Computer Name: GANDALF

Member of

Workgroup:

[✓ Domain:] SIMPLE

Create a Computer Account in the Domain

This option will create an account on the domain for this computer. You must specify a user account with the ability to add workstations to the specified domain above.

User Name:

Password:

Figure 6-5. Configuring a Windows NT client for domain logons

Like Windows 95/98, if NT complains that you are already logged in, you probably have an active connection to a share in the workgroup (such as a mapped network drive). Disconnect the resource temporarily by right-clicking on its icon and choosing the Disconnect pop-up menu item.

After you press the OK button, Windows should present you with a small dialog box welcoming you to the domain. At this point, you will need to reset the Windows NT machine. Once it comes up again, the machine will automatically present you with a log on screen similar to the one for Windows 95/98 clients. You can now log in using any account that you have already on the Samba server that is configured to accept logins.

Be sure to select the correct domain in the Windows NT logon dialog box. Once selected, it may take a moment for Windows NT to build the list of available domains.

After you enter the password, Windows NT should consult the primary domain controller (Samba) to see if the password is correct. Again, you can check the log files if you want to see this in action. If it worked, you have successfully configured Samba to act as a domain controller for Windows NT machines.

Domain Options

Table 6-9 shows the options that are commonly used in association with domain logons.

Table 6-9. Windows 95/98 Domain Logon Options

Option	Parameters	Function	Default	Scope
`domain logons`	boolean	Indicates whether Windows domain logons are to be used.	`no`	Global
`domain group map`	string (fully-qualified pathname)	Name of the file used to map Unix to Windows NT domain groups.	None	Global
`domain user map`	string (fully-qualified pathname)	Name of the file used to map Unix to Windows NT domain users.	None	Global
`local group map`	string (fully-qualified pathname)	Name of the file used to map Unix to Windows NT local groups.	None	Global
`revalidate`	boolean	If `yes`, Samba forces users to authenticate themselves with each connection to a share.	`no`	Share

domain logons

This option configures Samba to accept domain logons as a primary domain controller. When a client successfully logs on to the domain, Samba will return a special token to the client that allows the client to access domain shares without consulting the PDC again for authentication. Note that the Samba machine must be in user-level security (`security = user`) and must be the PDC in order for this option to function. In addition, Windows machines will expect a `[netlogon]` share to exist on the Samba server (see the section “Configuring Samba for Windows Domain Logons,” earlier in this chapter).

domain group map

This option specifies the location of a mapping file designed to translate Windows NT domain group names to Unix group names. The file should reside on the Samba server. For example:

```
/usr/local/samba/private/groups.mapping
```

The file has a simple format:

```
UnixGroup = NTGroup
```

An example is:

```
admin = Administrative
```

The specified Unix group should be a valid group in the */etc/group* file. The NT group should be the name to which you want the Unix group to map on an NT client. This option will work only with Windows NT clients.

domain user map

This option specifies the location of a mapping file designed to translate Unix usernames to Windows NT domain usernames. The file should reside on the Samba server. For example:

```
/usr/local/samba/private/domainuser.mapping
```

The file has a simple format:

```
UnixUsername = [\\Domain\\]NTUserName
```

An example entry is:

```
joe = Joseph Miller
```

The Unix name specified should be a valid username in the */etc/passwd* file. The NT name should be the username to which you want to Unix username to map on an NT client. This option will work with Windows NT clients only.

If you would like more information on how Windows NT uses domain usernames and local groups, we recommend Eric Pearce’s *Windows NT in a Nutshell*, published by O’Reilly.

local group map

This option specifies the location of a mapping file designed to translate Windows NT local group names to Unix group names. Local group names include those such as Administrator and Users. The file should reside on the Samba server. For example:

```
/usr/local/samba/private/localgroup.mapping
```

The file has a simple format:

```
UnixGroup = [BUILTIN\]NTGroup
```

An example entry is:

```
root = BUILTIN\Administrators
```

This option will work with Windows NT clients only. For more information, see Eric Pearce's *Windows NT in a Nutshell* (O'Reilly).

revalidate

This share-level option tells Samba to force users to authenticate with passwords each time they connect to a different share on a machine, no matter what level of security is in place on the Samba server. The default value is `no`, which allows users to be trusted once they successfully authenticate themselves. You can override it as:

```
revalidate = yes
```

You can use this option to increase security on your system. However, you should weigh it against the inconvenience of having users revalidate themselves to every share.

Logon Scripts

Samba supports the execution of Windows logon scripts, which are scripts (.BAT or .CMD) that are executed on the client when a user logs on to a Windows domain. Note that these scripts are stored on the Unix side, but are transported across the network to the client side and executed once a user logs on. These scripts are invaluable for dynamically setting up network configurations for users when they log on. The downside is that because they run on Windows, they must use the Windows network configuration commands.

If you would like more information on NET commands, we recommend the following O'Reilly handbooks: *Windows NT in a Nutshell*, *Windows 95 in a Nutshell*, and *Windows 98 in a Nutshell.*

You can instruct Samba to use a logon script with the `logon script` option, as follows:

```
[global]
    domain logons = yes
    security = user
    workgroup = SIMPLE

    os level = 34
    local master = yes
    preferred master = yes
    domain master = yes
    logon script = %U.bat

[netlogon]
    comment = The domain logon service
    path = /export/samba/logon
    public = no
    writeable = no
    browsable = no
```

Note that this example uses the `%U` variable, which will individualize the script based on the user that is logging in. It is common to customize logon scripts based on the user or machine name that is logging onto the domain. These scripts can then be used to configure individual settings for users or clients.

Each logon script should be stored at the base of the `[netlogon]` share. For example, if the base of the `[netlogon]` share is */export/samba/logon* and the logon script is *jeff.bat*, the file should be located at */export/samba/logon/jeff.bat*. When a user logs on to a domain that contains a startup script, he or she will see a small dialog that informs them that the script is executing, as well as any output the script generates in an MS-DOS-like box.

One warning: because these scripts are loaded by Windows and executed on the Windows side, they must consist of DOS formatted carriage-return/linefeed characters instead of Unix carriage returns. It's best to use a DOS- or Windows-based editor to create them.

Here is an example of a logon script that sets the current time to match that of the Samba server and maps two network drives, `h` and `i`, to individual shares on the server:

```
#  Reset the current time to that shown by the server.
#  We must have the "time server = yes" option in the
#  smb.conf for this to work.

echo Setting Current Time...
net time \\hydra /set /yes

#  Here we map network drives to shares on the Samba
#  server
```

```
echo Mapping Network Drives to Samba Server Hydra...
net use h: \\hydra\data
net use i: \\hydra\network
```

Roaming profiles

In Windows 95 and NT, each user can have his or her own *profile*. A profile bundles information such as: the appearance of a user's desktop, the applications that appear on the start menus, the background, and other miscellaneous items. If the profile is stored on a local disk, it's called a *local profile*, since it describes what a user's environment is like on one machine. If the profile is stored on a server, on the other hand, the user can download the same profile to any client machine that is connected to the server. The latter is called a *roaming profile* because the user can roam around from machine to machine and still use the same profile. This makes it particularly convenient when someone might be logging in from his or her desk one day and from a portable in the field the next. Figure 6-6 illustrates local and roaming profiles.

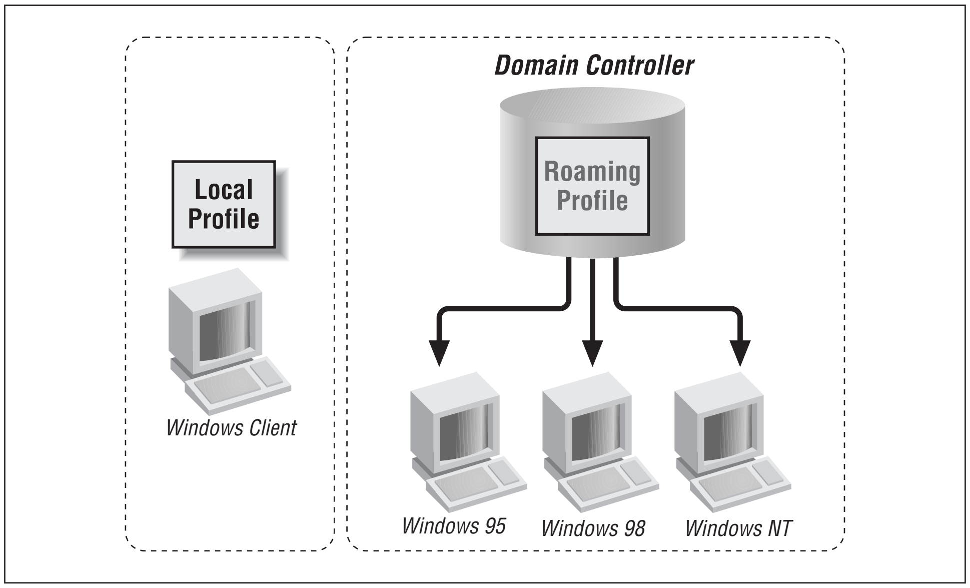


Figure 6-6. Local profiles versus roaming profiles

Samba will provide roaming profiles if it is configured for domain logons and you provide a tree of directories pointed to by the `logon path` option. This option is typically used with one of the user variables, as shown in this example:

```
[global]
    domain logons = yes
    security = user
    workgroup = SIMPLE
```

```
        os level = 34
        local master = yes
        preferred master = yes
        domain master = yes

        logon path =  \\hydra\profile\%U
```

We need to create a new share to support the profiles, which is a basic disk share accessible only by the Samba process' user (`root`). This share must be writeable, but should not be browseable. In addition, we must create a directory for each user who wishes to log on (based on how we specified our `logon path` in the example above), which is accessible only by that user. For an added measure of security, we use the `directory mode` and `create mode` options to keep anyone who connects to it from viewing or altering the files created in those directories:

```
    [profile]
      comment = User profiles
      path = /export/samba/profile
      create mode = 0600
      directory mode = 0700
      writable = yes
      browsable = no
```

Once a user initially logs on, the Windows client will create a *user.dat* or *ntuser.dat* file—depending on which operating system the client is running. The client then uploads the contents of the desktop, the Start Menu, the Network Neighborhood, and the programs folders in individual folders in the directory. When the user subsequently logs on, those contents will be downloaded from the server and activated for the client machine with which the user is logging on. When he or she logs off, those contents will be uploaded back on the server until the next time the user connects. If you look at the directory listing of a profile folder, you'll see the following:

```
    # ls -al

    total 321
    drwxrwxr-x   9 root  simple    Jul 21 20:44 .
    drwxrwxr-x   4 root  simple    Jul 22 14:32 ..
    drwxrwx---   3 fred  develope  Jul 12 07:15 Application Data
    drwxrwx---   3 fred  develope  Jul 12 07:15 Start Menu
    drwxrwx---   2 fred  develope  Jul 12 07:15 cookies
    drwxrwx---   2 fred  develope  Jul 12 07:15 desktop
    drwxrwx---   7 fred  develope  Jul 12 07:15 history
    drwxrwx---   2 fred  develope  Jul 12 07:15 nethood
    drwxrwx---   2 fred  develope  Jul 19 21:05 recent
    -rw-------   1 fred  develope  Jul 21 21:59 user.dat
```

The *user.dat* files are binary configuration files, created automatically by Windows. They can be edited with the Profile Editor on a Windows client, but they can be somewhat tricky to get correct. Samba supports them correctly for all clients up to NT 5.0 beta, but they're still relatively new.

Hints and HOWTOs for handling logon scripts are available in the Samba documentation tree, in both *docs/textdocs/DOMAIN.txt* and *docs/textdocs/PROFILES.txt.*

Mandatory profiles

Users can also have *mandatory profiles*, which are roaming profiles that they cannot change. For example, with a mandatory profile, if a user adds a command to the Start Menu on Tuesday, it will be gone when he or she logs in again on Wednesday. The mandatory profile is simply a *user.dat* file that has been renamed to *user.man* and made read-only on the Unix server. It normally contains settings that the administrator wishes to ensure the user always executes. For example, if an administrator wants to create a fixed user configuration, he or she can do the following:

1. Create the read-write directory on the Samba server.
2. Set the `logon path` option in the *smb.conf* file to point to this directory.
3. Logon as the user from Windows 95/98 to have the client populate the directory.
4. Rename the resulting *user.dat* to *user.man.*
5. Make the directory and its contents read only.

Mandatory profiles are fairly unusual. Roaming profiles, on the other hand, are one of the more desirable features of Windows that Samba can support.

Logon Script Options

Table 6-10 summarizes the options commonly used in association with Windows domain logon scripts.

Table 6-10. Logon Script Options

Option	Parameters	Function	Default	Scope
`logon script`	string (DOS path)	Name of DOS/NT batch file	None	Global
`logon path`	string (UNC server and share name)	Location of roaming profile for user	`\\%N\%U\profile`	Global

Table 6-10. Logon Script Options (continued)

Option	Parameters	Function	Default	Scope
`logon drive`	string (drive letter)	Specifies the logon drive for a home directory (NT only)	`Z:`	Global
`logon home`	string (UNC server and share name)	Specifies a location for home directories for clients logging on to the domain	`\\%N\%U`	Global

logon script

This option specifies a Windows .BAT or .CMD file with lines ending in carriage-return/line feed that will be executed on the client after a user has logged on to the domain. Each logon script should be stored at the base of a share entitled `[netlogin]` (see the section "Configuring Samba for Windows Domain Logons" for details.) This option frequently uses the `%U` or `%m` variables (user or NetBIOS name) to point to an individual script. For example:

```
logon script = %U.bat
```

will execute a script based on the username located at the base of the `[netlogin]` share. If the user who is connecting is `fred` and the path of the `[netlogin]` share maps to the directory */export/samba/netlogin*, the script should be */export/samba/netlogin/fred.bat*. Because these scripts are downloaded to the client and executed on the Windows side, they must consist of DOS formatted carriage-return/linefeed characters instead of Unix carriage returns.

logon path

This option provides a location for roaming profiles. When the user logs on, a roaming profile will be downloaded from the server to the client and activated for the user who is logging on. When the user logs off, those contents will be uploaded back on the server until the next time the user connects.

It is often more secure to create a separate share exclusively for storing user profiles:

```
logon path =  \\hydra\profile\%U
```

For more informaiton on this option, see the section "Logon Scripts," earlier in this chapter.

logon drive

This option specifies the drive letter on an NT client to which the home directory specified with the `logon home` option will be mapped. Note that this option will work with Windows NT clients only. For example:

```
logon home = I:
```

You should always use drive letters that will not conflict with fixed drives on the client machine. The default is Z:, which is a good choice because it is as far away from A:, C:, and D: as possible.

logon home

This option specifies the location of a user's home directory for use by the DOS NET commands. For example, to specify a home directory as a share on a Samba server, use the following:

```
logon home = \\hydra\%U
```

Note that this works nicely with the `[homes]` service, although you can specify any directory you wish. Home directories can be mapped with a logon script using the following command:

```
NET USE I: /HOME
```

In addition, you can use the User Environment Profile under User Properties in the Windows NT User Manager to verify that the home directory has automatically been set.

Other Connection Scripts

After a user successfully makes a connection to any Samba share, you may want the Samba server to execute a program on its side to prepare the share for use. Samba allows scripts to be executed before and after someone connects to a share. You do not need to be using Windows domains to take advantage of the options. Table 6-11 introduces some of the configuration options provided for setting up users.

Table 6-11. Connection Script Options

Option	Parameters	Function	Default	Scope
`root preexec`	string (Unix command)	Sets a command to run as `root`, before connecting to the share.	None	Share
`preexec (exec)`	string (Unix command)	Sets a Unix command to run as the user before connecting to the share.	None	Share
`postexec`	string (Unix command)	Sets a Unix command to run as the user after disconnecting from the share.	None	Share
`root postexec`	string (Unix command)	Sets a Unix command to run as `root` after disconnecting from the share.	None	Share

root preexec

The first form of the logon command is called `root preexec`. This option specifies a Unix command as its value that will be run *as the root user* before any connection to a share is completed. You should use this option specifically for performing actions that require root privilege. For example, `root preexec` can be used to mount CD-ROMs for a share that makes them available to the clients, or to create necessary directories. If no `root preexec` option is specified, there is no default action. Here is an example of how you can use the command to mount a CD-ROM:

```
[homes]
    browseable = no
    writeable = yes
    root preexec = /etc/mount /dev/cdrom2
```

Remember that these commands will be run as the root user. Therefore, in order to ensure security, users should never be able to modify the target of the `root preexec` command.

preexec

The next option run before logon is the `preexec` option, sometimes just called `exec`. This is an ordinary unprivileged command run by Samba as the user specified by the variable `%u`. For example, a common use of this option is to perform logging, such as the following:

```
[homes]
preexec = echo "%u connected to %S from %m (%I)\" >>/tmp/.log
```

Be warned that any information the command sends to standard output will not be seen by the user, but is instead thrown away. If you intend to use a `preexec` script, you should ensure that it will run correctly before having Samba invoke it.

postexec

Once the user disconnects from the share, the command specified with `postexec` is run as the user on the Samba server to do any necessary cleanup. This option is essentially the same as the `preexec` option. Again, remember that the command is run as the user represented by `%u` and any information sent to standard output will be ignored.

root postexec

Following the `postexec` option, the `root postexec` command is run, if one has been specified. Again, this option specifies a Unix command as its value that will be run *as the root user* before disconnecting from a share. You should use this option specifically for performing actions that require root privilege.

Working with NIS and NFS

Finally, Samba has the ability to work with NIS and NIS+. If there is more than one file server, and each runs Samba, it may be desirable to have the SMB client connect to the server whose disks actually house the user's home directory. It isn't normally a good idea to ship files across the network once via NFS to a Samba server, only to be sent across the network once again to the client via SMB. (For one thing, it's slow—about 30 percent of normal Samba speed). Therefore, there are a pair of options to tell Samba that NIS knows the name of the right server and indicate in which NIS map the information lives.

Table 6-12 introduces some of the other configuration options specifically for setting up users.

Table 6-12. NIS Options

Option	Parameters	Function	Default	Scope
`nis homedir`	boolean	If `yes`, use NIS instead of */etc/passwd* to look up the path of a user's home directory	`no`	Global
`homedir map`	string (NIS map name)	Sets the NIS map to use to look up a user's home directory	None	Global

nis homedir and homedir map

The `nis homedir` and `homedir map` options are for Samba servers on network sites where Unix home directories are provided using NFS, the automounter, and NIS (Yellow Pages).

The `nis homedir` option indicates that the home directory server for the user needs to be looked up in NIS. The `homedir map` option tells Samba what NIS map to look in for the server that has the user's home directory. The server needs to be a Samba server, so the client can do an SMB connect to it, and the other Samba servers need to have NIS installed so they can do the lookup.

For example, if user `joe` asks for a share called `[joe]`, and the `nis homedir` option is set to `yes`, Samba will look in the file specified by `homedir map` for a home directory for `joe`. If it finds one, Samba will return the associated machine name to the client. The client will then try to connect to *that* machine and get the share from there. Enabling NIS lookups looks like the following:

```
[globals]
    nis homedir = yes
    homedir map = amd.map
```