LINUX/WINDOWS INTEGRATION

Authentication and Authorisation in Samba

PART 2 Authentication and authorisation are the Tweedledum and Tweedledee of access control. They're separate, but you never see them apart, says Dr Chris Brown.

Actually, it's hard to over-estimate the importance of *Samba* in facilitating Linux / MS Windows integration and the adoption of Linux into the commercial marketplace. The quality of the work being done by the *Samba* development team certainly deserves our recognition and support. Last month, we began our tour of *Samba* by looking at the overall picture of file sharing in Windows. We talked about workgroups and shares, and about the SMB/CIFS protocol that makes it all work. We saw that *Samba* is a suite of programs and services that can provide file and print sharing services to Windows clients, and took a brief look at its configuration file, smb.conf. We learned about *Samba*'s web-based graphical configuration tool, SWAT (*Samba* Web Administration Tool). We also saw that *Samba* has client-side components that allow you, for example, to mount shares exported by Windows file servers onto a local Linux directory.

This month, we'll complete the tour by looking at the business of authentication and authorisation in *Samba*. Let's start by making sure we know what 'authentication' and 'authorisation' mean.

Authentication is the business of making certain that you know the identity of a user who is requesting a service - in this case, the service in question is access to a file share or a printer exported by a *Samba* server. From the end user's viewpoint, authentication is almost always just a matter of supplying a username and a password. Authorisation is the business of deciding whether a specific user is allowed to access a specific resource; for example, is user jane allowed to access the share she's trying to connect to, and if so, is she allowed to read and write the files in that share?

Authentication and authorisation usually go together. Authorisation checks against a user identity are meaningless unless you're sure you know who the user is. Conversely, authentication is a bit pointless unless you're subsequently going to use your knowledge of the user's identity to control what he can do. Nonetheless, it's important to remember that authentication and authorisation are actually separate operations. *Samba* always needs to have a Linux UID to run with, corresponding to the user on the windows client.

Author info

Dr Chris Brown is a UNIX and Linux consultant, trainer and writer of many years standing, though he claims to have spent most of them sitting down. He recently added an RHCE to his list of qualifications and reckons that, apart from taking his car for its MOT, it was the most stressful thing he's done for decades. Chris can be contacted at: chris@idlearn.com

Behind the scenes

It will help us understand *Samba*'s security mechanisms if we know a little bit about how it works behind the scenes. When *Samba*'s smb server starts up, there is initially just a single parent process running as root. When a user on a Windows machine connects to a *Samba* share, a new TCP connection is opened to the *Samba* server. The server will authenticate the user and check the user identity against the access control directives in its configuration file. If the user is allowed to connect to the share, *Samba* starts up a child process to service that connection. When the client accesses a file in the share, the child process switches to run under the identity of the client user. Requests to read and write any files within the share will then be checked against the usual 'rwx' file permissions for that user. These checks aren't made by *Samba*, they're just Linux's underlying security model being applied. The overall process, including some details we'll get to in a minute, is shown in **Fig1** on the right.

There are a couple of points to note about all this. First of all, access controls are applied at two levels; first by *Samba*, as determined by entries in its own config files, and second, by the underlying Linux filesystem. An important consequence of this is that *Samba* needs to establish an identity for the user which is known to Linux, corresponding either to an entry in /etc/passwd, or the NIS passwd map, or whatever user account database the system is configured to use. To put it in its most simple terms, *Samba* always needs to have a numeric Linux user ID (UID) corresponding to the client user.

Why is this an issue? With NFS-based file sharing (to digress for a moment) Linux simply assumes that the UID of the user on the server is the same as his UID on the client. So if user mike on a Linux client has the UID 507, NFS uses the UID 507 on the server, too. In the case of SMB-based file sharing, the client machine is probably some version of Windows, and of course, Windows simply doesn't use UNIX UIDs. So in this case, it's the user name that's transferred to the server – hence the need to be able to resolve the name onto a UID.

Encrypted passwords

Now a bit of history. Early versions of Windows, such as Windows 95, didn't really know about users at all. You didn't have to log in on the machine. It's true that if the machine was configured to use Microsoft networking, you'd be asked for a username and password when the machine booted. But it didn't use that information to decide whether you were allowed to use the machine, it simply squirrelled it away to use later if the machine tried to connect to an SMB share. Then it simply passed the name and password, in cleartext, to the server. In the case of a *Samba* server, it could use these to authenticate against a regular UNIX account, thus establishing a UID for the client.

Later versions of Windows, such as NT, 2000, and XP, mandate a proper authenticated login to the machine. More importantly, Microsoft changed the authentication scheme in SMB to use a challenge-response mechanism based on an encrypted password. Of course, the encryption method was not compatible with the one used in UNIX. Whilst this was a good step forward for

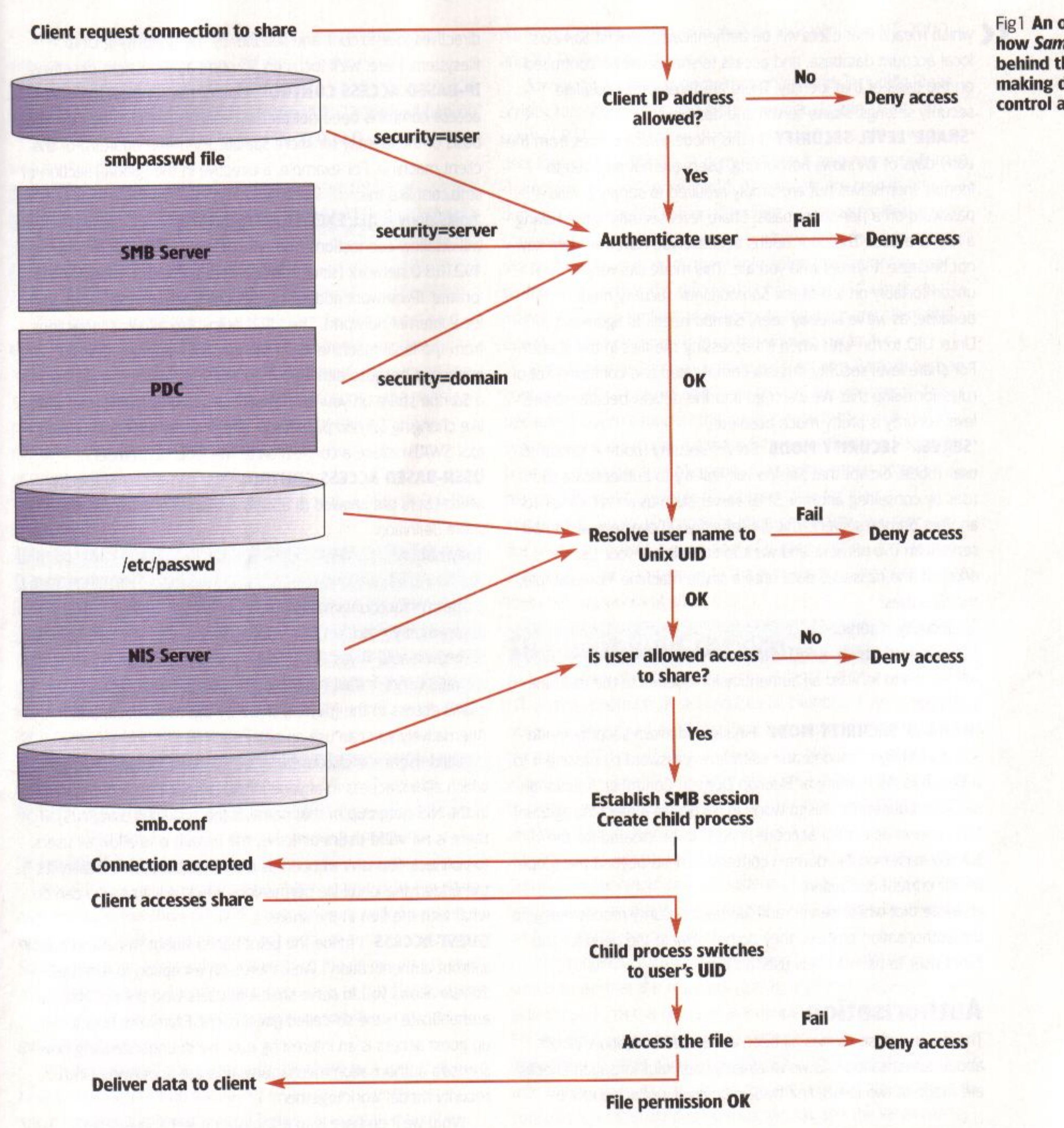

Fig1 **An overview of how *Samba* works behind the scenes in making decisions to control access.**

Windows, it caused consternation in the *Samba* world because the server no longer received a cleartext password and was thus not able to authenticate against the UNIX account database.

Samba can still be made to use cleartext passwords by including the directive

```
encrypted passwords = no
```

in the smb.conf file. It even provides patch files to doctor the registry on some versions of Windows to revert to the use of cleartext passwords. But cleartext passwords are not really a good idea, and this approach is, as they say, deprecated.

The Samba Password Database

To cope with the encrypted password mechanism, *Samba* was forced to use its own account database in which the windows user names and encrypted passwords could be stored. On my Red Hat system, this file is /etc/samba/smbpasswd, though if you've installed *Samba* from source, it might be elsewhere. There's also a program called *smbpasswd* that is provided as part of the *Samba* suite to manage this file. This program can be invoked by a normal user to change their *Samba* password, in much the same way that the **passwd** command is used to change regular Linux passwords.

The superuser can run *smbpasswd* to change any user's password or to add new users to the file. For example, the command:

```
# smbpasswd -a kim
```

will add the user kim to *smbpasswd*. It will also prompt for, and set, kim's password. For this command to succeed, kim must already have a regular Linux account.

The four security modes

To round off our discussion of authentication we should discuss the four 'security modes' of *Samba*, as set by the 'security' directive in the [global] section of smb.conf. Typically, you will see a line like this:

```
security = user
```

»

« which means that users will be authenticated against *Samba*'s local account database, and access to shares will be controlled on the basis of that identity. There are three other possible security settings: share, server, and domain.

'SHARE' LEVEL SECURITY In this mode, which comes from the early days of Windows networking, users are not required to identify themselves but are simply required to supply a valid password on a per-share basis. Share level security is like having a key to a door. The door opens because you have the right key, not because it knows who you are. This mode sits very uncomfortably on top of the *Samba*/Linux security model because, as we've already seen, *Samba* needs to figure out a Linux UID to run with when it's accessing the files in the share. For share level security, it has a complicated and confusing set of rules for doing this. We won't go into the details, because share level security is pretty much obsolete.

'SERVER' SECURITY MODE Server security mode is similar to user mode, except that *Samba* will first try to authenticate the user by consulting another SMB server such as an NT server (or another *Samba* server). This is useful if you have several *Samba* servers on the network and want to consolidate your user account and password data onto a single machine. For example, the directives:

```
security = server
password server = SATURN
```

tell *Samba* to forward all authentication queries to the machine SATURN.

'DOMAIN' SECURITY MODE Finally, in domain security mode *Samba* will try to validate the username/password by passing it to a Windows NT Primary or Backup Domain Controller. Additional setup is required for this to work. In particular, a machine account (also known as a 'trust account') needs to be created for the *Samba* server on the domain controller. This is beyond the scope of our current discussion.

Note that whilst 'server' and 'domain' security modes delegate the authorisation process, they do not obviate the need for the client user to have a Linux user account.

Authorisation

The discussion so far has all been about authentication. What about authorisation? As we've already seen, authorisation checks are made at two levels, first by *Samba* itself, under control of directives in smb.conf, and second, by the underlying Linux filesystem. Here, we'll focus on *Samba*'s authorisation directives.

IP-BASED ACCESS CONTROL One of the simplest forms of access control is done not on the basis of user identity but on the basis of the identity (or more specifically the IP address) of the client machine. For example, a directive in the [global] section of smb.conf like this:

```
hosts deny = ALL EXCEPT 192.168.0. 127.
```

will disallow connections from all hosts except those on the 192.168.0 network (since this is one of those non-routable 'private' IP network addresses, this entry presumably refers to the local internal network). The "127." entry also allows connections from the local machine using the loopback address. This is important because although it's unlikely you'd want to connect to a *Samba* share on your own machine, some administrative tasks like changing *Samba* passwords, and running the configuration tool SWAT, require a connection to the local *Samba* server.

USER-BASED ACCESS CONTROL You can explicitly specify which users are allowed to access a share. For example, the share definition:

```
[payroll]

  path = /accounts/payroll
  read only = no
  browseable = yes
  valid users = david mary
```

allows access to the [payroll] share by the users david and mary. Alternatively you can use an entry such as

```
  valid users = @accountants
```

which allows access to anyone in the Linux group accountants (or in the NIS netgroup of that name, if the system is using NIS). If there is no **valid users** directive, the default is to allow *all* users to connect. This isn't as open as it sounds, because in reality it's the underlying Linux file permissions which will limit who can do what with the files in the share.

GUEST ACCESS I made the point earlier about "no authorisation without authentication". Well, there's an exception to this rule. *Samba* allows you to serve shares to users who are not able to authenticate – the so-called guest users. Examining how to set up guest access is an interesting exercise in understanding how *Samba*'s authorisation mechanisms and the underlying Linux security model work together.

What we'll do here is to establish a guest share called 'blackboard' corresponding to the directory /home/bboard. Now, even for unauthenticated guests, *Samba* must have a Linux user identity to adopt when accessing the share. In this case we'll create a 'mythical' user called sam (Mythical in the sense that there is no warm pink body of this name; and no-one can actually log in to Linux as sam. The account exists simply to give *Samba* an identity to use when it is accessing the share on behalf of guests.)

First, we'll create the Linux user account like this:

```
# useradd -c "Samba guest" -d /home/bboard -s /sbin/nologin sam
```

Specifying a 'shell' of /sbin/nologin ensures that no-one can log in to this account; consequently, sam's password is essentially irrelevant. (Guest logins aside, it's worth pointing out that if you're creating a user account on Linux for the sole purpose of allowing access to shares through *Samba*, rather than (for example) allowing the user to log in and type commands at a shell, it's a good idea to disable the login as shown above.)

Samba resources

Useful links for more information

Samba is probably one of the best documented Open Source projects around. The man page for smb.conf is extremely detailed, and is also available as part of the SWAT package as an HTML file (smb.conf.5.html). There's also an extensive collection of HOWTO pages, collected together as a single PDF file. On a Red Hat system it's in /usr/share/doc/samba-xxx/docs, where xxx is the version number. All of this, and more, is also available online, at the main *Samba* web site www.samba.org. Just follow the 'documentation' link. You can also download source and binaries of *Samba* itself from this site.

If you prefer something you can read in a hammock, I'd strongly recommend O'Reilly's *Using Samba* by Ts, Eckstein and Collier-Brown (ISBN 0-5960-0256-4). Not only is this one of the most authoritative books on *Samba*, it's also one of the most current, with coverage of *Samba* 2.2 and the upcoming version 3.0. Make sure you get the second edition published February 2003 though – the original is getting a bit dated.

Finally, I was intending to provide a link to Daniel Robbins' excellent series of *Samba* articles that were on IBM's developerworks web site, but I just checked, and it seems they are there no more. This site does, however, offer a wealth of quality, technical tutorials and you might want to check it out. Go to the www.ibm.com/developerworks site and follow the 'Linux' link.

The useradd command will also create the /home/bboard directory and set it to be owned by sam. It looks like this:

```
# ls -ld /home/bboard
drwx------ 3 sam   sam   4096 Jun 15 11:27 /home/bboard/
```

Now we need to add two lines to the **[global]** section of smb.conf, like this:

```
[global]
  ...
  map to guest = Bad User
  guest account = sam
  ...
```

The first line tells *Samba* that if a user tries to connect who does not have an account on the server, it should map the identity of the user to that of the guest account. The second line specifies the identity to use for the guest account – the mythical user 'sam' we just created. (As for all the directives, there is a compiled-in default for this; in this case it's usually the user account 'nobody'.)

Finally, we will be needing a section in smb.conf to define the share itself:

```
[blackboard]
  path = /home/bboard
  guest ok = yes
  writeable = yes
```

The **guest ok** line means just what it says – it's OK to allow guest users to connect to this share. The last line makes the share writeable – probably not something you'd really want to do for a guest share in the real world.

After making these changes, you'll want to tell *Samba* to re-read the file. You can do this by sending a SIGHUP signal:

```
# killall -HUP smbd
```

or, on Red Hat, by using the handy **service** command:

```
# service smb reload
```

Now, if all is well, you should be able to connect to the share as a guest. To test this I created a user account called sara on a Windows 2000 client. Sara does not exist as an account on my *Samba* machine. I was able to connect to the share, and to read and write files there. Interestingly, if you look at the files that sara created, you'll see that they're actually owned by our mythical guest user, sam.

What happens if you connect to this share as a user that *does* have a valid account on the server (*ie* a user with entries in /etc/passwd and in *smbpasswd*)? To test this I created another user mike with accounts both on Windows 2000 and on the *Samba* server. Whilst mike is able to see the blackboard share, he won't be able to list the files in it. Why is this? In this case, *Samba* is happy to allow mike to connect to the share, but the underlying **rwx------** permissions on the /home/bboard directory won't allow him to access the file. In this case, *Samba* is now running as 'mike' not as 'sam'.

Fancy footwork with Samba

This introductory tutorial has focussed on the use of *Samba* as a file server and describes functionality which has, for the most part, been around in *Samba* for quite a while (since version 2.0). Version 2.2 added and consolidated several other capabilities which allow *Samba* to take over a number of additional roles in Microsoft networks. Principally, *Samba* can act as a PDC (Primary Domain Controller). This includes the authentication of Windows 95, 98, NT, 2000 and XP users against its security database. Note, however, that Windows 95, 98 and XP home edition don't perform a full domain login in the way that Windows NT, 2000 and XP Professional clients do.

Samba also supports domain logon services including logon scripts, roaming profiles and system policies. Logon scripts are scripts (.BAT or .CMD files) that are executed on a client when a user logs on to a domain. A common use is to automatically map shares onto network drive letters. The scripts are held in the domain controller, but run on the client. Roaming profiles are somewhat inappropriately named – it's the user that can roam; the profile stays put, that's the whole point. Profiles define user preferences such as the desktop configuration and what's on the menus. Roaming profiles allow users to log in on any client and download their own profile from the domain controller.

One thing that *Samba* won't do is to act as a Backup domain controller (BDC) against an NT-based PDC (or *vice versa*). This is because it doesn't support the protocol which Microsoft use to synchronise the security database between PDCs and BDCs. You have to remember that Microsoft don't publish their protocols, nor do they make available the source code for their implementation. Writing code to interoperate with Windows on the network involves a good deal of packet-sniffing and reverse engineering. It's a testament to the hard work of the *Samba* team that anything works at all!

The winbindd service

As we've seen, although *Samba* can delegate authorisation to an NT domain controller, it still requires all clients to have valid Linux accounts. These can be in the local /etc/passwd file, or be served from the NIS *passwd* map, if you're using NIS. Although NIS allows you to centralise the user account database, you still need to create user accounts for each of your Windows client users, and with a large user population this is a chore system administrators would prefer to avoid. A fairly recent addition to *Samba*, the *winbindd* daemon, extends the range of mechanisms which Linux can use to perform user name lookups by allowing an NT server to be consulted.

This requires changes to the entries in the name service switch file so that the resolvers (the library functions which actually perform the user name lookups) know where to look. For example, an entry in /etc/nsswitch.conf like this:

```
passwd:     files winbind
```

tells the resolvers to first look in the local *passwd* file and then to consult the *windbindd* daemon. For details, see the *winbindd* manual page.

On the horizon

At the time of writing this tutorial, the beta release of *Samba* 3.0 has just been announced. By the time you get to read this, a production release may be out. The authentication system has been almost completely re-written in this release. There is support for Active Directory (according to the release notes, "this release is able to join a ADS realm as a member server and authenticate users using LDAP/Kerberos"). However, *Samba* 3.0 cannot function as an Active Directory domain controller, and the restriction of not being able to synchronise between NT PDCs and BDCs remains. There is improved support for UNICODE (16-bit character sets). There's a new 'net' command, designed to look like the net command in Windows/DOS, and a slew of other improvements. This release looks set to move *Samba* a good deal further down the road to provide a complete Linux/Windows integration solution. LXF