HOUR 13

Troubleshooting Techniques

I find it interesting that this hour is the first hour after the half-way point of this book—if you are counting hours. It is also the last hour before "Part IV—Going Production."

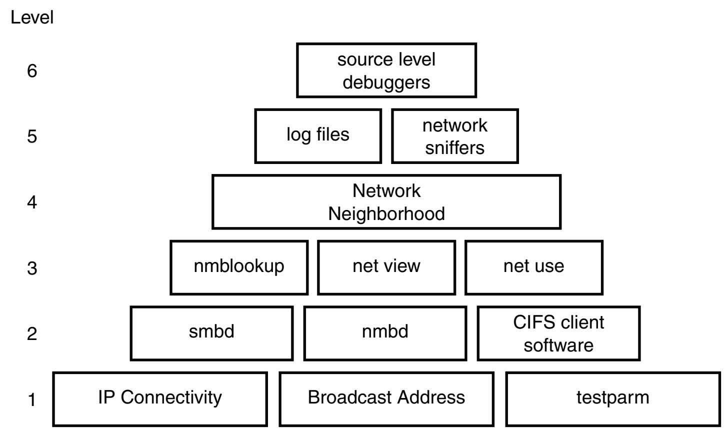
Why have I waited so long to introduce techniques for troubleshooting Samba? I believe that troubleshooting is a chicken-and-egg type of problem. When you need to know how to troubleshoot a software package, you don't have enough information to know what to do on your own. However, once you have this information, you don't necessarily need a troubleshooting guide because you can pinpoint the problems based on your own experiences. The problem with writing about methods of tracking down and solving broken Samba configurations is that you must have a general understanding of how things should work before you can understand why the techniques are valid. But, as I said before, once you have this knowledge, you can usually figure things out for yourself. It is a vicious cycle, really.

The approach we will take in this hour is to identify a collection of best practices for Samba administrators. These guidelines will be in the form of common error messages and their meanings as well as techniques and avenues for obtaining help when you are absolutely stuck. In the tradition of Unix and Open Source/Free Software, this hour will help you to stand on the shoulders of those who have already been in the position of fixing broken servers and have succeeded.

Egypt, Samba, and Bugs

Wouldn't it be horrible to replace the engine in your car, only to find out later that it was simply out of gas? This may seem like a silly question, but I have seen too many network administrators waste time because they did not check for basic problems first. With this in mind, Figure 13.1 presents a pyramid for analyzing Samba problems.

FIGURE 13.1
Analyzing problems with Samba from the bottom up.



The topics presented at the bottom of the pyramid are fundamental to the ones at the upper levels. It is no wonder why Windows clients cannot access a file share on your Samba box if the server is unplugged from the network. Though this representation of troubleshooting requirements doesn't always hold to be true (you can always view log files), it is better to start from the bottom when diagnosing a problem. As we climb higher, involvement with Samba internals grows, ending with the heights (or depths, depending on your point of view) of stepping through the source code line by line using a debugger.

Available Tools at Our Disposal

When searching for the root of a problem, there are three basic resources available to us:

- Existing documentation, such as this book, HOWTO's distributed with Samba itself, and mailing list archives
- Interactive communication with other administrators and users, via means such as IRC, newsgroups, and mailing lists
- Software tools, such as the utilities included with Samba and network monitoring programs

We will spend the majority of the hour focusing on personal troubleshooting efforts. This does not imply that the first two items in this list are not as important. Extra documentation and the advice of experienced administrators can be invaluable. This book could not contain the wealth of knowledge possessed by the Samba community. Who knows? You might even have a chance to return the favor to someone else one day.

Documentation

Much work has gone into cleaning up the documentation for Samba 2.2, although there is still much to be done. One of the major changes was the conversion from a markup language called YODL (*Yet oneOther Document Language*) to SGML/DocBook. One of the products of this is the Samba-HOWTO-Collection book. This collection of tutorials and explanations is available in PDF and HTML format, both of which are included with the Samba distribution. The HTML version is linked from the main page when logging into SWAT.

There are many other individual text files stored in the `docs/textdocs` directory that have yet to be updated and converted. One that receives a great deal of use is the `DIAGNOSIS.txt` file. This file is the original troubleshooting guide for Samba. Although it is fairly old now, it is still the first guide many administrators use when trying to track down a problem.

People

The Samba community of people can be one of the best resources for tracking down problems. You must remember, though, that all the people whom you might contact on the various mailings and newsgroups help because they want to and not because they are paid to. This includes many of those who develop Samba, commonly referred to as the Samba Team. This means that the ultimate burden for fixing your problems still falls on your own shoulders.

When posting to or answering questions on any mailing list or newsgroup, you should follow common Internet etiquette (or Netiquette). If you do not, you will find that people will be less than helpful. However, if you are considerate in your postings, someone will normally respond.

You can find out more information about the Samba mailing lists and how to join them at `http://lists.samba.org/`. Some of the available mailing lists are

- `samba@samba.org`—This is the main Samba mailing list for general information related to installing, configuring, and maintaining Samba servers.
- `samba-technical@samba.org`—This is the mailing list for discussions regarding the development of Samba. If you feel like pitching in, join the list, open up `vi`, and start working through the source code.
- `samba-bugs@samba.org`—This address is not a mailing list, but rather an address for reporting actual bugs in the Samba applications. This is not an address to be used for general help questions.

Many people have found the interactive nature of IRC channels to be more conducive for debugging problems in real time. There are two popular channels hosted by `irc.openprojects.net`. The `#samba-technical` channel often contains Samba Team members and other developers discussing current plans and the future of Samba. The `#samba` channel can be used when seeking help with configuration problems.

The Usenet newsgroup `comp.protocols.smb` is another good source of information about configuring and testing Samba.

The Test Environment

In order to properly address the troubleshooting methods presented in the remainder of the hour, we must make some statements about environment. These tests assume that

- The Samba server named `POGO` is located at IP address `192.168.1.75` using a netmask of `255.255.255.0`. These are specific details that relate to the underlying operating system on which Samba is running.
- Both `smbd` and `nmbd` are configured to start as daemons. Experience has proven that this configuration is much less problematic than having them launched by `[x]inetd` upon demand.
- Our Windows client (95, 98, ME, NT, 2000 or XP) is named `WIN-CLIENT`.
- The Windows system has only the CIFS client (i.e. "Client for Microsoft Networks" or `Workstation` service), a network adapter, and the TCP/IP protocol installed. The client is using address `192.168.1.135` with a netmask of

255.255.255.0. You can refer to Hours 10 and 11 for details on configuring Windows clients.

- Both WIN-CLIENT and POGO are located on the same logical IP network, meaning that a broadcast packet from one machine can be seen by the other.
- Both WIN-CLIENT and POGO are members of the workgroup STY-SAMBA.

The Samba server is using the following smb.conf:

```
[global]
     netbios name = POGO
     workgroup = STY-SAMBA
     security = user
     encrypt passwords = yes

[public]
     path = /tmp
     read only = no
```

With these components in place, we are ready to begin our troubleshooting journey.

Level 1: Beginning the Climb up the Pyramid

At the base of the pyramid of Figure 13.1, three components are presented as having foundational importance. These should be the first items we check if a problem arises.

- General TCP/IP connectivity
- Matching network masks and broadcast addresses among servers and clients
- A working smb.conf

Pinging the Server and Client

One of the basic tools for verifying TCP/IP connectivity is the ping command. At the risk of oversimplifying the ICMP protocol, ping sends a request to a host and asks, “Are you alive?” If the host does not respond, the machine sending the ping request assumes that it is not connected to the network or not currently available (for example, powered off).

First, we will attempt to ping the client from our Unix server.

```
$ ping win-client
PING win-client (192.168.1.135) from 192.168.1.74 : 56(84) bytes of data.
64 bytes from win-client (192.168.1.135): icmp_seq=0 ttl=255 time=2.138 msec
64 bytes from win-client (192.168.1.135): icmp_seq=1 ttl=255 time=2.181 msec
64 bytes from win-client (192.168.1.135): icmp_seq=2 ttl=255 time=2.263 msec

--- pogo.plainjoe.org ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/mdev = 2.138/2.194/2.263/0.051 ms
```

The output from `ping` will appear slightly different depending upon the operating system installed, but success or failure should be obvious. The previous example was from a RedHat 7.1 host.

One of the fundamental services of TCP/IP networks is the Domain Name Service (DNS). If the server is unable to resolve the hostname to an address, you will see a message similar to

```
$ ping win-client
ping: unknown host win-client
```

If this occurs, the first step is to try the `ping` command again, but using the IP address of the client.

```
$ ping 192.168.1.135
```

If this succeeds, we can point a finger at the DNS configuration. The most common errors are

- The DNS configuration on the server (`/etc/resolv.conf`) is broken
- The `win-client` name does not contain an entry in the DNS zone
- The DNS server is currently unreachable

If we are unable to `ping` the IP address of the client, the next step is to verify that the network card of both the client and the server is installed correctly and functioning properly. You should also ensure that all the network cabling and intermediate components, such as hubs and switches, are connected properly.

Describing how to debug hardware components of either Windows or Unix systems can be a very detailed subject. I am of the opinion that experience is really the best teacher in this case. If you suspect hardware failures, try to find someone who is already familiar with your configuration and ask for advice. Books and other documentation are always helpful, but they are rarely as quick to provide an answer as an experienced administrator.

The `ping` tool is also available on Windows clients. If you cannot find the `ping.exe` executable on the Windows client, make sure that the TCP/IP protocol is listed in the installed network components, which are displayed by the Network Control Panel applet.

Windows' `ping` command produces output similar to the Linux `ping` utility.

```
C:\WINDOWS>ping pogo

Pinging pogo [192.168.1.75] with 32 bytes of data:

Reply from 192.168.1.75: bytes=32 time=19ms TTL=255
Reply from 192.168.1.75: bytes=32 time=3ms TTL=255
Reply from 192.168.1.75: bytes=32 time=15ms TTL=255
Reply from 192.168.1.75: bytes=32 time=6ms TTL=255

Ping statistics for 192.168.1.75:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 3ms, Maximum =  19ms, Average =  10ms
```

The process of diagnosing a failed `ping` on the Windows client is the same as the one previously described for a Unix server.

Comparing Network Broadcast Addresses

It is possible to `ping` one host from another and still have a misconfigured netmask and broadcast address for the host's network interface. Why would a broken broadcast address matter to Samba?

Remember that our discussion of NetBIOS so far has covered only broadcast-based name registration and resolution. The NetBIOS name services are crucial to features such as the Network Neighborhood and connecting to a shared directory or printer. In Hour 18, "WINS and NetBIOS Name Services," we will see how the NetBIOS name space can be extended beyond a single network segment. Until then we must rely on using broadcast packets to locate NetBIOS clients and servers.

Both `POGO` and `WIN-CLIENT` should be using a network mask of `255.255.255.0` and a broadcast address of `192.168.255`. We can verify this information on our Unix server by running the `ifconfig` command with a single argument of the network adapter's name.

```
$ /sbin/ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:04:5A:0C:1C:19
          inet addr:192.168.1.75  Bcast:192.168.255.255  Mask:255.255.0.0
          inet6 addr: fe80::204:5aff:fe0c:1c19/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:68006 errors:0 dropped:0 overruns:0 frame:0
          TX packets:100783 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:12186135 (11.6 Mb)  TX bytes:121642120 (116.0 Mb)
          Interrupt:3 Base address:0x100
```

Our broadcast address is `192.168.255.255`, which is incorrect. When run as root, `ifconfig` can be used to set the broadcast address.

```
root# ifconfig eth0 192.168.1.75 netmask 255.255.255.0 broadcast 192.168.1.255
```

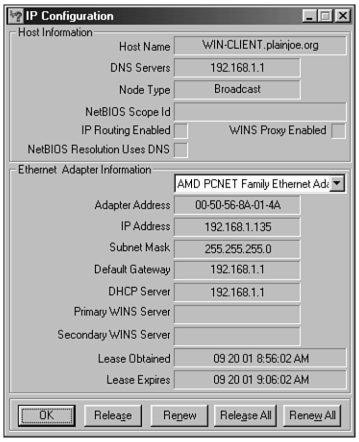
Network settings made from a command line are almost always lost when the network interface is reset or the system is rebooted. Refer to your server's documentation for the correct procedure to make this change persistent.

Viewing the current TCP/IP settings on a Windows NT/2000 client can be done with the `ipconfig.exe` tool. The `/all` option instructs `ipconfig` to display detailed information about all installed adapters. This listing is from a Windows NT 4.0 client named `CAESAR`, which obtains its IP settings via DHCP.

```
C:\WINNT\>ipconfig /all

Windows NT IP Configuration

        Host Name . . . . . . . . . : caesar.plainjoe.org
        DNS Servers . . . . . . . . : 192.168.1.1
        Node Type . . . . . . . . . : Broadcast
        NetBIOS Scope ID. . . . . . :
        IP Routing Enabled. . . . . : No
        WINS Proxy Enabled. . . . . : No
        NetBIOS Resolution Uses DNS : No

Ethernet adapter AMDPCN1:

        Description . . . . . . . . : AMD PCNET Family Ethernet Adapter
        Physical Address. . . . . . : 00-50-56-91-01-4A
        DHCP Enabled. . . . . . . . : Yes
        IP Address. . . . . . . . . : 192.168.1.134
        Subnet Mask . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . : 192.168.1.1
        DHCP Server . . . . . . . . : 192.168.1.1
        Lease Obtained. . . . . . . : Thursday, September 20, 2001 1:53:08 PM
        Lease Expires . . . . . . . : Thursday, September 20, 2001 2:03:08 PM
```

Windows 9x/ME systems do not include a command line tool for viewing local TCP/IP settings. These hosts use a graphical program named `winipcfg.exe`. The IP configuration for a Windows 98 client is shown in Figure 13.2.

If either the network mask or the broadcast address is incorrect on this client, refer to the instructions in Hours 10 and 11 on how to set the correct value. If the client obtained its information via DHCP, you should refer to your DHCP server's documentation for possible solutions.



IP Configuration

Host Information

Host Name	WIN-CLIENT.plainjoe.org
DNS Servers	192.168.1.1
Node Type	Broadcast
NetBIOS Scope Id	
IP Routing Enabled	
WINS Proxy Enabled	
NetBIOS Resolution Uses DNS	

Ethernet Adapter Information

AMD PCNET Family Ethernet Ada

Adapter Address	00-50-56-8A-01-4A
IP Address	192.168.1.135
Subnet Mask	255.255.255.0
Default Gateway	192.168.1.1
DHCP Server	192.168.1.1
Primary WINS Server	
Secondary WINS Server	
Lease Obtained	09 20 01 8:56:02 AM
Lease Expires	09 20 01 9:06:02 AM

FIGURE 13.2
Viewing the Windows IP configuration using the `winipcfg.exe` *utility.*

Verifying the `smb.conf` Settings

Because Samba uses an extremely large amount of parameters in `smb.conf`, developers have provided a command line tool for verifying the syntax of a configuration file. The utility, named `testparm`, has already been described in Hour 4, “Starting Your Feet to Dance.” It is being mentioned here again so that we do not overlook simple `smb.conf` errors when hunting for the source of a problem.

`testparm` can be instructed to parse a specific configuration file by using the `-s` switch. It is a good idea to run the check on a new configuration file before putting it into production.

```
$ testparm -s /usr/local/samba/lib/smb.conf.new
Load smb config files from /usr/local/samba/lib/smb.conf.new
Processing section "[public]"
Loaded services file OK.
# Global parameters
[global]
        coding system =
        client code page = 850
        code page directory = /usr/local/samba/lib/codepages
<...remaining output deleted...>
```

After reviewing the specified configuration file, `testparm` continues to print a version of `smb.conf` that contains all parameter values, including default ones. This can help to verify that `smbd` and `nmbd` are using the values that you expect.

Because default values can change between releases, it is important to use the version of `testparm` that matches the `smbd` and `nmbd` daemons installed on your server.

Level 2: Local Server and Client Software

The second level of Figure 13.1 turns to the server and client software. Our goal is to make sure that both machines are running and responding to NetBIOS and CIFS requests. For the most part, we are concerned with an isolated host. It is not until the third layer that the server and client will begin to carry on a conversation.

smbd

For this test, `smbd` must be running, so first we check whether this is the case by using the `ps` command. The actual `ps` arguments on your system may be different than in this example from a Linux server.

```
$ ps -ef | grep smbd
root     28592     1  0 12:37 ?  00:00:00 /usr/local/samba/bin/smbd -D
```

Having a running `nmbd` is not a requirement of the tests in this section.

After verifying that `smbd` is running, or after launching it, if necessary, our next test is to enumerate the list of shares and browse masters known by the server. In Hour 4, `smbclient`'s capability to browse a server was first introduced as a means of testing our new Samba server. The `-L <server>` option instructs `smbclient` to list the available resources on the server. The `-N` flag (anonymous login) is used to temporarily avoid any potential problems with authentication. This step should be executed while logged on to the Samba server locally (that is, not from another Unix host on the network).

```
$ smbclient -L pogo -N
added interface ip=192.168.1.75 bcast=192.168.1.255 nmask=255.255.255.0
Anonymous login successful
Domain=[STY-SAMBA] OS=[Unix] Server=[Samba 2.2.2]

        Sharename      Type      Comment
        ---------      ----      -------
        public         Disk
        IPC$           IPC       IPC Service (Samba 2.2.2)
        ADMIN$         Disk      IPC Service (Samba 2.2.2)

        Server               Comment
        ---------            -------
        POGO                 Samba 2.2.2

        Workgroup            Master
        ---------            -------
        STY-SAMBA            POGO
```

Two common problems can result in a failure of this test. The first error,

```
error connecting to 192.168.1.75:139 (Connection refused)
Connection to <server> failed
```

is caused by `smbd` not running or not being able to bind to port 139. An inability to bind to the correct port can be caused by configuring `smbd` to start from `[x]inetd` (possibly left over from a previous Samba installation) and then attempting to launch the server as a daemon. The most common means of correcting this is to ensure that `smbd` can actually start. Because `smbd` does not print error messages to the console window, it is a good idea to view the last few lines of the associated log file (see Level 5 for more information on log files).

The second error message often seen by administrators is

```
session request to <server> failed (Not listening for calling name)
```

When connecting locally using `smbclient`, this error is almost always a result of a misconfigured `hosts allow` or `hosts deny` parameter in `smb.conf`. The server is running by rejecting the NetBIOS session setup. These two parameters are covered in Hour 17, "Security Tips," when we focus on Samba network security concerns.

From the description of the NetBIOS Name Service in Hour 2, it sounds like we used the wrong NetBIOS name when connecting to the server. However, this is not the case here. This error cannot be caused by a broken `nmbd` installation because `nmbd` does not even have to be running currently.

Assuming that we can successfully enumerate shares, we can next test Samba's ability to authenticate users. The details of connecting to a disk share or printer service using `smbclient` were covered thoroughly in the previous hour when we explored CIFS clients that are available for Unix hosts. In this test, we will attempt to connect to the `[public]` share as the account named "`user1`" with a password of "`secret`".

```
$ smbclient //pogo/public -U user1%secret
added interface ip=192.168.1.75 bcast=192.168.1.255 nmask=255.255.255.0
Domain=[STY-SAMBA] OS=[Unix] Server=[Samba 2.2.2]
smb: \>
```

If smbd is able to successfully authenticate the login name/password pair and that user is authorized to access the requested share, we are greeted with an `smb: \>` prompt.

If Samba is not able to validate the user's credentials, we are informed that

```
session setup failed: ERRSRV - ERRbadpw (Bad password - name/password
pair in a Tree Connect or Session Setup are invalid.)
```

There can be many reasons for this, such as a misspelled username or password, a missing smbpasswd entry for the user in the case of `encrypt passwords = yes`, or an invalid `guest account` if we are allowing non-authenticated access.

If the user was correctly authenticated, but could not access the request service, smbclient reports that the

```
tree connect failed: ERRDOS - ERRnosuchshare (You specified an
invalid share name)
```

This can be caused by a misspelled share name, permissions on the share which restrict the user in question from accessing that directory or printer, or a bad path statement in the share's definition as given by smb.conf.

nmbd

To test nmbd, we again use the ps command to ensure that it is running.

```
$ ps -ef | grep nmbd
root     29054     1  0 15:53 ?  00:00:00 /usr/local/samba/bin/bin/nmbd -D
```

If nmbd does not appear in the list reported by ps, it should be started as root using the normal means (`/usr/local/samba/bin/nmbd -D`).

Samba's nmblookup tool was briefly mentioned in Hour 4. Samba servers have a special NetBIOS name, __SAMBA__, to which they will always respond. By querying the server for this name, we can verify that nmbd is working correctly. The -U flag is used to specify the address to which the query should be sent.

```
$ ./nmblookup -U 127.0.0.1 __SAMBA__
querying __SAMBA__ on 127.0.0.1
192.168.1.75 __SAMBA__<00>
```

If nmbd had not been running, the query would have resulted in the following error message.

```
name_query failed to find name __SAMBA__
```

This can also be caused by not including the loopback interface name for the interfaces parameter in smb.conf and setting `bind interfaces only = yes`. Both of these parameters are discussed during Hour 23 in the context of capacity planning and system tuning.

Next we will check to see if nmbd was able to successfully register the name POGO.

```
$ nmblookup -U 127.0.0.1 POGO
querying POGO on 127.0.0.1
192.168.1.75 POGO<00>
```

Any error messages, such as “name query failed”, are most likely caused by the same conditions that would cause the query for the __SAMBA__ name to fail. Another possible reason for a failure is that the server was unable to register its NetBIOS name. If this is the case, you can locate the host that currently owns the name by sending a name query request to the broadcast address of the local subnet.

```
$ nmblookup -B 192.168.1.255 POGO
querying POGO on 192.168.1.255
192.168.1.98 POGO<00>
```

In this example, the name POGO has been registered by a host at address 192.168.1.98 and not our Samba server. Obviously, the way to correct this problem is to rename the rogue machine.

Windows’ NetBIOS Interface

The NetBIOS interface on the Windows machine must also be verified, just as we checked the Samba installation on the server. Because the CIFS client and NetBIOS interface are intimately linked together on Microsoft clients, it is enough to check that the latter component is functioning correctly.

Windows’ NetBIOS name query tool, nbtstat.exe, contains a few extra features beyond those available to nmblookup. One of these (-n) is the ability to ask the NetBIOS interface what names it has successfully registered.

```
C:\WINDOWS> nbtstat -n

Node IpAddress: [192.168.1.135] Scope Id: []
            NetBIOS Local Name Table

   Name               Type         Status
---------------------------------------------
WIN-CLIENT     <00>  UNIQUE      Registered
STY-SAMBA      <00>  GROUP       Registered
WIN-CLIENT     <03>  UNIQUE      Registered
```

If the “Client for Microsoft Networks” has not been installed, nbtstat.exe will report (assuming that the tool is installed on the system at all)

```
Failed to access NBT driver 1
```

A more subtle error is when the Windows client reports that it has registered a workgroup name, but not its unique workstation name as shown here.

```
Name              Type         Status
---------------------------------------------
STY-SAMBA      <00>  GROUP       Registered
```

This is often caused by a machine existing on the network with a duplicate NetBIOS name. The Windows client needs a unique name to use when establishing a NetBIOS Session with a server. Until the client is able to successfully register a workstation name, it will be unable to do things such as browse the Network Neighborhood or map a network drive.

Level 3: Remote Access to Shares

So far, we have ensured that both machines are accessible on the network and that the NetBIOS and CIFS software components are working locally. At this level of the pyramid, we move beyond the local server and client to test how well the two can communicate with each other.

Name Resolution

We will again turn to the `nmblookup` and `nbtstat.exe` programs for help in verifying that the server can resolve the name of the client and vice versa. This test will come in two stages. The first part will issue a broadcast name resolution request to test the responses of the server and the client. This is done by passing the network's broadcast address to `nmblookup` (`-B 192.168.1.255`) when querying for a name. This brings network communication into play. We will first attempt to resolve the server's name by running

```
$ nmblookup -B 192.168.1.255 pogo
querying pogo on 192.168.1.255
192.168.1.75 pogo<00>
```

Next we will query for the client's name using the same broadcast address.

```
$ nmblookup -B 192.168.1.255 win-client
querying win-client on 192.168.1.255
192.168.1.135 win-client<00>
```

If all has gone well up to this point, this test should seldom fail. However, if either step does result in an error, verify that the broadcast address on each client is set to the same value. You should also check for any `interfaces` or `bind interface only` settings that would prevent the Samba host from responding to queries arriving from this particular subnet.

Next we will perform a NetBIOS Node Status Lookup request from the server to the client and from the client to the server. This step sends a directed packet to the IP address given, requesting a list of all unique and group NetBIOS names registered by the host. We will begin by querying Samba from the Windows machine.

```
C:\WINDOWS\> nbtstat -A 192.168.1.75

       NetBIOS Remote Machine Name Table

   Name               Type         Status
---------------------------------------------
POGO           <00>  UNIQUE      Registered
POGO           <03>  UNIQUE      Registered
POGO           <20>  UNIQUE      Registered
..__MSBROWSE__.<01>  GROUP       Registered
STY-SAMBA      <00>  GROUP       Registered
STY-SAMBA      <1D>  UNIQUE      Registered
STY-SAMBA      <1E>  GROUP       Registered

MAC Address = 00-00-00-00-00-00
```

The `nmbd` daemon always reports a MAC address of `00-00-00-00-00-00` to node status requests.

We can perform the same operation on the Samba server to gain information about the client. The options for performing a Node Status Request with `nmblookup` are exactly the same as those used by Windows' `nbtstat.exe` tool.

```
$ nmblookup -A 192.168.1.135
Looking up status of 192.168.1.135
        WIN-CLIENT      <00> -         B <ACTIVE>
        STY-SAMBA       <00> - <GROUP> B <ACTIVE>
        WIN-CLIENT      <03> -         B <ACTIVE>
```

If either of these requests fails, you should back up to the IP connectivity tests of Level 1 and the NetBIOS interface checks for `nmbd` and the Windows client described in Level 2.

Enumerating Shares from the Windows Client

We have already used the `smbclient` tool to enumerate the list of file and printer services on our Samba server in Level 2 of the pyramid. During this section, we will perform the same test except that the request will come from the remote Windows client.

The `net.exe` command is Microsoft's Swiss army knife for its CIFS clients. This tool provides an equivalent version of the "`smbclient -L`" command. The `view` option can be used to browse the contents of a workgroup or, when given the name of a specific server (for example, `\\POGO`), enumerate the shared resources on that host.

Unless you have enabled the special guest settings described in Hour 7, "Security Levels and Passwords," it will be necessary to make sure the Samba server can successfully authenticate you. When executing `net view \\POGO`, the Windows client will attempt to connect to the server using the login name and password of the currently logged on user. If a connection already exists, such as a mapped network drive, the `net view` command will use that session. Do not be surprised if the preceding command results in an "`Access Denied`" error message or indicates that your password is incorrect.

A successful view of `POGO` appears as

```
C:\WINDOWS\>net view \\pogo
Shared resources at \\pogo

Samba [2.2.2]

Share name   Type         Used as  Comment

-------------------------------------------------------------------------------
public       Disk
The command completed successfully.
```

Connecting to a Share Remotely

This step is often more of a goal for administrators than it is a test. Here, we will test the inter-operability of the Windows client with our server.

Before continuing, you should remember that there are several issues concerning Windows clients and clear text passwords that require special attention. These were thoroughly covered in Hours 10 and 11. We have conveniently avoided those problems here by enabling encrypted passwords in `smb.conf`.

Assuming that we are logged in to the Windows console with a valid username and password, the following command will connect the `[public]` share on `POGO` to drive `P:` on the local client.

```
C:\WINDOWS\> net use p: \\pogo\public
The command completed successfully.
```

Windows 9x/ME will not allow you to use a login name when accessing a CIFS server that is different from the one specified when logging on to the Windows console. Windows NT 4.0/2000/XP supports an optional `/user:<username>` switch for setting the username to be used in the `SMBsessetup&X` request. The following variation, which connects to the server as `user1`, works only under Windows NT/2000/XP systems.

```
C:\WINNT\>net use \\pogo\public /user:user1
The password or user name is invalid for \\pogo\public.

Type the password for \\pogo\public:
The command completed successfully.
```

There are many more things that can break with authentication. Often, these problems can be located and solved only by digging through Samba's log files, which will be discussed later in this hour.

Level 4: Network Browsing

Network browsing (that is, the Network Neighborhood) is a rather complicated dance. So much so that we will spend two hours discussing how it works and how to integrate Samba with other CIFS clients and servers. If you have reached Level 4 of the pyramid and the Network Neighborhood is still empty or only half working, I would suggest that you look at Hours 18, 19, and 20. These hours will take you through the steps necessary to configure Samba as a good neighbor both in its local network segment and with machines located on a remote network.

Level 5: A Wealth of Information—Log Files and Packets

There are times when debugging subtle problems, which no form of diagnostic tool seems to give enough information to, tracks down the real issue. The first four levels of Figure 13.1 can be viewed as general diagnostics steps used to run a sanity check on the server when first installing or upgrading Samba. At Level 5, we have crossed over from the general diagnostics phase into the realm of hard-core Samba administration. Sooner or later, everyone hits a problem that requires cranking up the debug level, scanning the log files, and monitoring network traffic.

Samba's Log Files

Table 13.1 (originally presented as Table 5.2) describes the information recorded at each log level. The actual division of information is not this clean throughout all of Samba, so take the categories as a rule of thumb and not a firm design.

TABLE 13.1 Information Recorded at the Various Log Levels

Level	*Description*
0	Critical failures such as failing to open a log file, dropping a connection, or receiving an unknown CIFS command
1	Connection and session information
2–4	System administration debugging information
5–9	Moderate developer debugging data
10	Full developer debugging information

In Samba 2.0, the debug level of a running process could be incremented by sending it the `USR1` signal and decremented using the `USR2` signal. With the introduction of Samba's internal messaging system in 2.2, this responsibility has been handed over to the `smbcontrol` tool.

To query the current log level of an `smbd` (for example, `pid 1234`), we would send the `debuglevel` message to the process by executing the `smbcontrol` command as root.

```
root# smbcontrol 1234 debuglevel
Current debug level of PID 1234 is 0
```

If we needed to increase this to debug level 10, we would send the process a `debug` message such as

```
root# smbcontrol 1234 debug 10
root# smbcontrol 1234 debuglevel
Current debug level of PID 1234 is 10
```

It is often helpful to set `debug timestamp = no` at higher log levels. This prevents `smbd` from adding a timestamp header to each log entry, which can become distracting.

The next question is, "What do we do with all of this information that `smbd`, `nmbd`, or some other Samba tool has just recorded?"

Here is one example of how log files can help to locate the source of a problem. In this case, we are attempting to connect to a disk share from a Windows client. However, `smbd` will never accept the password that we enter for the connection. When testing the server using `smbclient` we receive the error

```
$ smbclient //pogo/public -U testuser%test
session setup failed: ERRSRV - ERRbadpw (Bad password - name/password
pair in a Tree Connect or Session Setup are invalid.)
```

We are sure that the `testuser` account has a valid entry in `smbpasswd` and that the password is set to the string "`test`". After making another attempt to connect the share with

```
log level = 10
log file = /usr/local/samba/var/log.%m
```

added to the `[global]` section of `smb.conf`, we notice the following lines in the `log.pogo`:

```
pdb_getsampwnam: search by name: testuser
startsmbfilepwent_internal: opening file /usr/local/samba/private/smbpasswd
getsmbfilepwent: returning passwd entry for user root, uid 0
getsmbfilepwent: returning passwd entry for user jerry, uid 786
getsmbfilepwent: returning passwd entry for user guest1, uid 782
getsmbfilepwent: returning passwd entry for user testuser, uid 791
endsmbfilepwent_internal: closed password file.
pdb_getsampwnam: found by name: testuser
build_sam_account: smbpasswd database is corrupt!  username testuser
     not in unix passwd database!
Couldn't find user 'testuser' in passdb.
```

The final line of this excerpt gives us a clue to our problem. Samba could not locate a Unix account for the `testuser` account. The reason for this is that someone has commented out its entry in the `/etc/passwd` file.

```
#testuser:x:791:100::/dev/null:/bin/false
```

After removing the hash mark (`#`) from the beginning of the account entry, we try one more time to connect using `smbclient`. This time, however, we are met with success.

```
$ smbclient //pogo/public -U testuser%test
Domain=[STY-SAMBA] OS=[Unix] Server=[Samba 2.2.2]
smb: \>
```

This is just one example of how Samba's log files can tell you where they are sick. Though the amount of data logged at high levels (for example, 10) can be intimidating, here are some key phrases to `grep` for when searching for a problem.

- `fail`
- `error`
- `unsuccessful`
- `corrupt`
- `unknown`

Unless you know the exact string you are searching for, use `grep`'s `-i` option to define the search string to be case insensitive (for example, `grep -i fail log.smbd`).

Monitoring Network Traffic

Many networks use shared media, such as Ethernet, to connect computers together. This is similar to a conference call, where all the connected participants can hear everything spoken by every other person on the line. Many conference calls have a passcode or special access number because of the sensitive nature of the conversation. This prevents uninvited people from randomly listening in. Networks work in much the same way. If anyone who wanted to view all network packets as they flew by on the wire were allowed to do so, invariably a password or credit card number would be seen.

Many large networks now use Ethernet switches in the place of hubs. A switch will allow you only to see packets sent to and from your host and any packets sent to your broadcast address.

Packet sniffers, tools that allow administrators to view network traffic as it passes by, can also be used as a diagnostic tool for determining why hosts or services cannot communicate with each other. We will quickly examine two such programs: Ethereal and Microsoft's Network Monitor. The first is freely available, while the second is commercial software. Our goal is to understand what these tools do and what type of information we can gain from them.

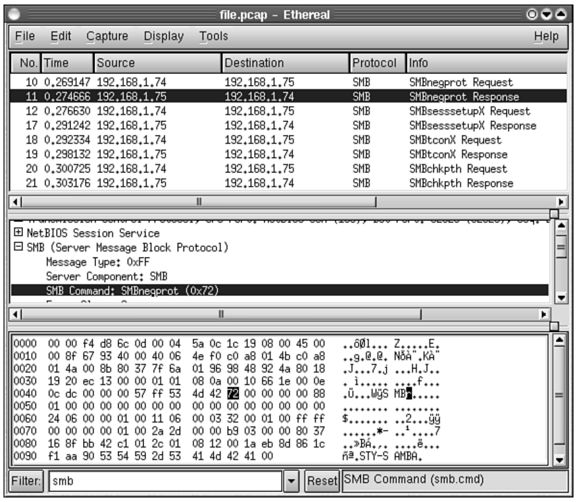
Ethereal

Ethereal is a GTK+ based program available for Unix and Windows clients from `http://www.ethereal.com/`. Its main window is displayed in Figure 13.3. One of the main advantages of Ethereal is its ability to decode packets into a more human-readable form.

Tethereal, a command line version of `Ethereal`, is also distributed with its GUI based sibling. To get a better idea of what `ethereal` (and `tethereal`) can tell us, we will watch the traffic between `smbclient` and our Samba server when connecting to a share.

```
$ smbclient //pogo/public -U user1%secret
```

FIGURE 13.3
Ethereal's main window for displaying captured packets.



```
No. Time     Source        Destination   Protocol Info
 10 0.269147 192.168.1.74  192.168.1.75  SMB      SMBnegprot Request
 11 0.274666 192.168.1.75  192.168.1.74  SMB      SMBnegprot Response
 12 0.276630 192.168.1.74  192.168.1.75  SMB      SMBsesssetupX Request
 17 0.291242 192.168.1.75  192.168.1.74  SMB      SMBsesssetupX Response
 18 0.292334 192.168.1.74  192.168.1.75  SMB      SMBtconX Request
 19 0.298132 192.168.1.75  192.168.1.74  SMB      SMBtconX Response
 20 0.300725 192.168.1.74  192.168.1.75  SMB      SMBchkpth Request
 21 0.303176 192.168.1.75  192.168.1.74  SMB      SMBchkpth Response
```

```
⊞ NetBIOS Session Service
⊟ SMB (Server Message Block Protocol)
    Message Type: 0xFF
    Server Component: SMB
    SMB Command: SMBnegprot (0x72)
```

```
0000  00 00 f4 d8 6c 0d 00 04  5a 0c 1c 19 08 00 45 00   ..öØl... Z.....E.
0010  00 8f 67 93 40 00 40 06  4e f0 c0 a8 01 4b c0 a8   ..g.@.@. NðÀ¨.KÀ¨
0020  01 4a 00 8b 80 37 7f 6a  01 96 98 48 92 4a 80 18   .J...7.j ...H.J..
0030  19 20 ec 13 00 00 01 01  08 0a 00 10 66 1e 00 0e   . ì..... ....f...
0040  0c dc 00 00 00 57 ff 53  4d 42 72 00 00 00 00 88   .Ü...WÿS MBr.....
0050  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0060  24 06 00 00 01 00 11 06  00 03 32 00 01 00 ff ff   $....... ..2...ÿÿ
0070  00 00 00 00 01 00 2a 2d  00 00 b9 03 00 00 80 37   ......*- ..¹....7
0080  16 8f bb 42 c1 01 2c 01  08 12 00 1a eb 8d 86 1c   ..»BÁ.,. ....ë...
0090  f1 aa 90 53 54 59 2d 53  41 4d 42 41 00            ñª.STY-S AMBA.
```

To capture the network traffic transmitted between `smbclient` and `smbd` running on the same host, we will instruct `tethereal` to watch the loopback network interface (`-i lo`) for all packets destined for `port 139`.

```
root# tethereal -i lo port 139
  0.268257        pogo -> pogo        SMB SMBnegprot Request
  0.269917        pogo -> pogo        SMB SMBnegprot Response
  0.272619        pogo -> pogo        SMB SMBsesssetupX Request
  0.280524        pogo -> pogo        SMB SMBsesssetupX Response
  0.281527        pogo -> pogo        SMB SMBtconX Request
  0.283225        pogo -> pogo        SMB SMBtconX Response
```

These packets illustrate the steps required to establish a connection to a CIFS share initially presented in Figure 2.8. It is possible to get `tethereal` to decode the packets and print a tree based view (-V) of each one. This is identical to the view provided by Ethereal. In this example, we are examining a portion of the Negotiate Protocol (`SMBnegprot`) response sent from `smbd` to `smbclient`.

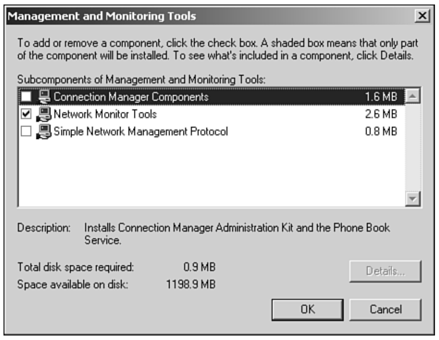
```
root# tethereal -i lo -V port 139
<...output deleted...>
SMB (Server Message Block Protocol)
    Message Type: 0xFF
    Server Component: SMB
    SMB Command: SMBnegprot (0x72)
    Error Class: Success
    Reserved: 0
    Error Code: No Error
```

```
<...output deleted...>
Security Mode: 0x03
    .... ...1 = Security  = User
    .... ..1. = Passwords = Encrypted
    .... .0.. = Security signatures not enabled
    .... 0... = Security signatures not required
<...output deleted...>
```

Can you guess what the values of the `security` and `encrypt passwords` parameters are based on in this listing? The "`Security Mode`" flag is an 8-bit field used to represent this information. The least significant bit (`0x01`) is set when the server is operating in user-mode security (that is, `security = [user|server|domain]`). The next bit (`0x02`) is 1 when the server supports NTLM (that is, `encrypt passwords = yes`).

Microsoft's Network Monitor

Microsoft packages a packet-tracing tool with the Windows NT/2000 Server CD-ROM and with the System Management Software (SMS) CD-ROM called Network Monitor (also known as `netmon`). `netmon` is composed of two parts: the tool itself and a local or remote agent to which it connects. Both must be installed for the software to function correctly. Figure 13.4 displays the installation window shown by a Windows 2000 Server.



Management and Monitoring Tools

To add or remove a component, click the check box. A shaded box means that only part of the component will be installed. To see what's included in a component, click Details.

Subcomponents of Management and Monitoring Tools:

☐	Connection Manager Components	1.6 MB
☑	Network Monitor Tools	2.6 MB
☐	Simple Network Management Protocol	0.8 MB

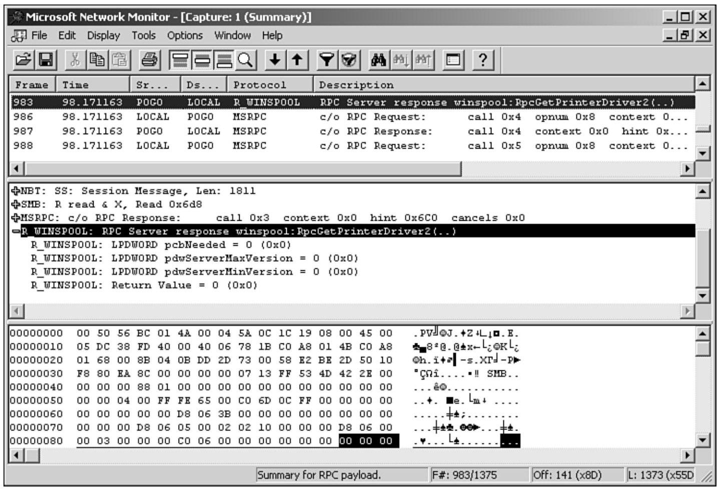
Description: Installs Connection Manager Administration Kit and the Phone Book Service.

Total disk space required: 0.9 MB
Space available on disk: 1198.9 MB

FIGURE 13.4
Installing the Network Monitor tools on Windows 2000.

There are two distinct versions of `netmon`, neither of which is freely available like Ethereal. The version included with the Windows NT/2000 Server CD-ROM allows only for viewing packets sent to and from the local machine, similar to an Ethernet switch. The version included with the SMS CD-ROM enables the network interface to be put into promiscuous mode, with which all packets on the shared media can be seen.

FIGURE 13.5
Decoding a printing MS-RPC response using `netmon`*.*



Microsoft Network Monitor - [Capture: 1 (Summary)]

```
Frame  Time       Sr...  Ds...  Protocol    Description
983    98.171163  POGO   LOCAL  R_WINSPOOL  RPC Server response winspool:RpcGetPrinterDriver2(..)
986    98.171163  LOCAL  POGO   MSRPC       c/o RPC Request:     call 0x4  opnum 0x8  context 0...
987    98.171163  POGO   LOCAL  MSRPC       c/o RPC Response:    call 0x4  context 0x0  hint 0x...
988    98.171163  LOCAL  POGO   MSRPC       c/o RPC Request:     call 0x5  opnum 0x8  context 0...
```

```
+NBT: SS: Session Message, Len: 1811
+SMB: R read & X, Read 0x6d8
+MSRPC: c/o RPC Response:     call 0x3  context 0x0  hint 0x6C0  cancels 0x0
-R_WINSPOOL: RPC Server response winspool:RpcGetPrinterDriver2(..)
   R_WINSPOOL: LPDWORD pcbNeeded = 0 (0x0)
   R_WINSPOOL: LPDWORD pdwServerMaxVersion = 0 (0x0)
   R_WINSPOOL: LPDWORD pdwServerMinVersion = 0 (0x0)
   R_WINSPOOL: Return Value = 0 (0x0)
```

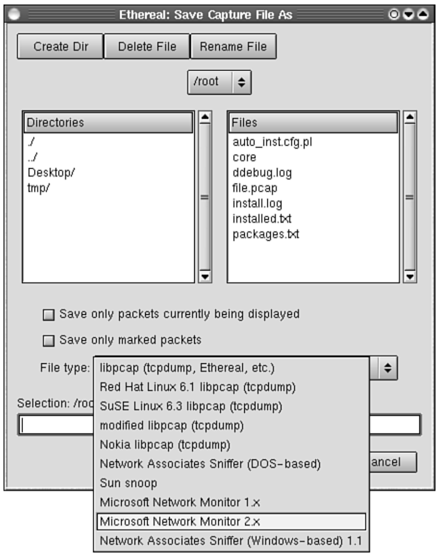
```
00000000  00 50 56 BC 01 4A 00 04 5A 0C 1C 19 08 00 45 00
00000010  05 DC 38 FD 40 00 40 06 78 1B C0 A8 01 4B C0 A8
00000020  01 68 00 8B 04 0B DD 2D 73 00 58 E2 BE 2D 50 10
00000030  F8 80 EA 8C 00 00 00 00 07 13 FF 53 4D 42 2E 00   SMB..
00000040  00 00 00 88 01 00 00 00 00 00 00 00 00 00 00 00
00000050  00 00 04 00 FF FE 65 00 C0 6D 0C FF 00 00 00 00
00000060  00 00 00 00 00 D8 06 3B 00 00 00 00 00 00 00 00
00000070  00 00 00 D8 06 05 00 02 02 10 00 00 00 D8 06 00
00000080  00 03 00 00 00 C0 06 00 00 00 00 00 00 00 00 00
```

Summary for RPC payload. | F#: 983/1375 | Off: 141 (x8D) | L: 1373 (x55D

As can be seen in Figure 13.5, `netmon` has the ability to decode packets, much like Ethereal. In some cases Microsoft's tool is able to do a better job because their engineers added parsers to decode undocumented protocols such as MS-RPC.

Ethereal has the ability to read and write Network Monitor's CAP file format (see Figure 13.6). So though you may not have the full version of `netmon` available without purchasing SMS, it is possible to take advantage of its decoding capabilities by capturing the traffic using Ethereal and converting the packet trace to Network Monitor's own format.

FIGURE 13.6
Saving a network trace captured by Ethereal to `netmon`*'s CAP format.*



Ethereal: Save Capture File As

/root

Directories: ./ ../ Desktop/ tmp/

Files: auto_inst.cfg.pl, core, ddebug.log, file.pcap, install.log, installed.txt, packages.txt

☐ Save only packets currently being displayed
☐ Save only marked packets

File type:
- libpcap (tcpdump, Ethereal, etc.)
- Red Hat Linux 6.1 libpcap (tcpdump)
- SuSE Linux 6.3 libpcap (tcpdump)
- modified libpcap (tcpdump)
- Nokia libpcap (tcpdump)
- Network Associates Sniffer (DOS-based)
- Sun snoop
- Microsoft Network Monitor 1.x
- Microsoft Network Monitor 2.x
- Network Associates Sniffer (Windows-based) 1.1

Selection: /roo

Level 6: Samba Internals

One of the advantages of Open Source/Free Software is the availability of the source code. Many people, however, never make use of this. During this section, we will quickly look at means of tracking down bugs in the Samba source code using a symbolic debugger. You will need to be comfortable with programming C and somewhat familiar with Unix development tools such as `gcc` and `gdb` to get the most possible out of this section. If you do not feel quite up to speed with your programming skills or have no interest in fixing bugs in the source code, this section can be skipped.

For any type of source level debugging, it is imperative that Samba be compiled to include debugging symbols. With this extra information, we can match the currently running instruction with the exact line in the source code that generated that instruction.

Enabling debugging symbols in the Samba binaries can be done by setting the `--enable-debug` flag when running the `./configure` script. This switch is compatible with all other `./configure` options, so we can use it in conjunction with Samba's support for PAM, for example.

```
root# ./configure --enable-debug  --with-pam
```

The rest of Samba's installation process remains the same.

The `--enable-debug` option sets the `-g` flag when compiling. This will normally result in much larger binaries (as much as 4 to 8 times in size), which is something to consider when installing Samba in tight places with a minimal amount of free disk space.

There are two types of bugs we will be concerned with here: those bugs that make Samba crash, and those that make Samba do the wrong thing.

Fortunately, the first class of bugs is not found very often in stable Samba releases. If Samba does encounter a fatal error such as a segmentation fault (attempting to access a region of memory that was off limits, generally due to an invalid pointer), a message similar to the following will be written to the `smbd` log file.

```
===============================================================
INTERNAL ERROR: Signal 11 in pid 16124 (2.2.0)
Please read the file BUGS.txt in the distribution
===============================================================
```

Before shutting down, however, `smbd` will also execute a global `panic action` command if one is defined. By default, the `panic action` parameter has no value. Because

Windows clients often reconnect transparently to the user, these crashes can easily be missed. An obvious `panic action` setting would be to immediately attach to the faulty `smbd` with `gdb` and examine where the problem occurred. The following command will launch an `xterm` on the local server's display and attach to the crashed `smbd`.

```
panic action = /usr/X11R6/bin/xterm -display :0 \
  -T Panic -n Panic -e /bin/sh -c 'gdb /usr/local/samba/bin/smbd %d'
```

Few servers run `X` in order to conserve resources. In this case, we can have the `smbd` sleep and wait for us to manually attach.

```
panic action = "/bin/sleep 99999"
```

We can then attach to the errant `smbd` at our leisure.

The `panic action` parameter is provided as an option for developers. All of the latest Samba code is available online for access via CVS (Concurrent Versions System). More information about Samba's CVS repository can be found at `http://samba.org/samba/cvs.html`. CVS clients for many platforms can be downloaded from `http://cvshome.org`.

The second class of bugs can be much harder to track down. The reason for this is that the CIFS protocol is very complex and messy. Locating exactly where the problem lies in Samba's source code can be compared to looking for a needle in a haystack. One tool that can help you to navigate through the functions and data structures is the Cygnus Source Navigator. This Integrated Development Environment (IDE) has been released under the GPL and can be downloaded from `http://sources.redhat.com/sourcenav/`. Source Navigator is an excellent tool for understanding any piece of software for which the source code is available.

Summary

Troubleshooting any problem is somewhere between an art and a science. This hour has provided a methodology for tracking down problems with Samba installations through a process of elimination. The steps have been divided up into layers, each of which depends on the successful passing of the tests of the previous level. The troubleshooting layers shown in Figure 13.1 from the bottom up are

- Level 1—General network connectivity tests and a working `smb.conf`
- Level 2—Local client and server software
- Level 3—Remote access to shares

Samba24Hc13.qxd 2001-11-07 19:13 Page 240

- Level 4—Network browsing
- Level 5—Logging and monitoring network traffic
- Level 6—Source level debugging

Various mailing lists, IRC channels, and Usenet newsgroups can put you in touch with people who can help you with the art of troubleshooting. Remember to approach the problem from all possible angles, not simply focusing on the piece that is broken.

Q&A

Q Are there archives for the different Samba mailing lists that I can search to see whether anyone has ever asked my question before?

A Yes. There is a searchable archive for all the mailing lists served by `samba.org`. See `http://samba.org/samba/archives.html` for details.

Q Where can I find out information about commercial support for Samba?

A The main Samba Web site includes a page containing a list of companies that offer commercial support for Samba. Look under the “Support” page. Vendors are grouped by country.

Q What information should I include when reporting a problem or bug in Samba?

A At the very minimum, you should include

- The server’s operating system and relevant version numbers (for example, kernel 2.4.9 running on a SuSE 7.2 system, Solaris 8 x86, and so forth)
- The version of Samba (or branch tag and date of the cvs snapshot) you are running
- The operating system version, including hot fixes and Service Packs, of the clients that are experiencing the problem
- A good description of the problem and the means to reproduce it, if possible

You should also be prepared to make your `smb.conf` and log files available if necessary.

Q Is there a current list of known bugs available for released versions of Samba?

A The Samba developers maintain a list of known bugs and their status at `http://samba.org/samba/buglist.html`.

New Terms

packet sniffer This is a common name for a class of network tools, either software or hardware, that are able to display the raw data being transmitted across a network. These utilities are also called network tracers or packet tracers. Some also provide the capability to parse the packets and display the information in a format more readable to humans.