Abstract

Last Update : Fri Oct 10 00:59:58 CEST 2003

This book is a collection of documents that might be useful for people developing samba or those interested in doing so. It's nothing more than a collection of documents written by samba developers about the internals of various parts of samba and the SMB protocol. It's still (and will always be) incomplete. The most recent version of this document can be found at <http://devel.samba.org/>.

ATTRIBUTION

CONTENTS

Contents

Chapter 1

NETBIOS IN A UNIX WORLD

1.1 Introduction

level" functions. It supports only 8 (so far) of the SMBtrans sub-functions. Even NT doesn't support them all.

Chapter 2

<

5.

NotForIdemp: 0x20

NotforBcast: 0x40

NoUuid: 0x80

UINT16[]

2.2.3.13 POL_HND (LSA policy handle)

UINT64 logon ID

UNIHDR user name unicode header

UNIHDR workgroup name unicode header

char[16] arc4 LM OWF Password

char[16] arc4 NT OWF Password

UNISTR2 domain name unicode string

UNISTR2 user name unicode string

UNISTR2 workstation name unicode string

2.2.3.22 SAM_INFO (sam logon/logo id info structure)

Section

DOM_SID[num_sids] other domain SIDs?

2.2.3.27 SH_INFO_1_PTR 59 10.156 -37.545 Td [(2.2.3.27)-1100(SH)]TJETq1 0 0 1 220.0

SV_TYPE_LOCAL_LIST

with the pipe name `\PIPE\srvsvc`

UINT16 maxtsize maximum transmission fragment size (0x1630)

UINT16 maxrsize max receive fragment size (0x1630)

UINT32 assocgid associated group id (0x0)

UINT32 numelements the number of elements (0x1)

UINT16 contextid presentation context identi er (0x0)

UINT8 numsyntaxes pr esu472er [(prealw g oa9091yd [(pre Tf -12en [(pre1?)32(n)32(textid)]TJ0 g 0 G/P

2.3.2.8 RPC_ResNorm RW

UINT32 allochint # size of the stub data in bytes

UINT16 prescontext # presentation context identi er (same as request)

UINT8 cancelcount # cancel count? (0x0)

UINT8 reserved # 0 - one byte padding

* stub USE TvPacket # the remainder of the reply

2.3.3 Tail

The end of each of the NTLSA and NETLOGON named pipes ends with:

.....

Establish a connection to the IPC$ share (SMBtconX). use encrypted passwords.
Open an RPC Pipe with the name "*nn*PIPE*nn*lsarpc". Store the le handle.
Using the le handle, send a Set Named Pipe Handle state to 0x4300.
Send an LSA Open Policy request. Store the Policy Handle.
Using the Policy Handle, send LSA Query Info Policy requests, etc.
Using the Policy Handle, send an LSA Close.
Close the IPC$ share.

De nes for this pipe, identifying the query are:

LSA Open Policy: 0x2c

LSA Query Info Policy: 0x07

0x07 Enumerate T

LSA OpenSretcy:

fe2A

OBJ_ATTR object attributes

UINT32 1 - desired access

2.3.6.2 Response

POL_HND LSA policy handle

return 0 - indicates success

2.3.7 LSA Query Info Policy

Note: The info class in response must be the same as that in the request.

2.3.7.1 Request

POL_HND LSA policy handle

UINT16 info class (also a policy handle?)

2.3.7.2 Response

VOID* undocumented bu er pointer

UINT16 info class (same as info class in request).

```
}

return    0 - indicates success
```

VOID*[num_entries] undocumented domain SID pointers to be looked up.
DOM_SID[num_entries] domain SIDs to be looked up.

char[16] completely undocumented 16 bytes.

2.3.11.2 Response

DOM_REF domain reference response

UINT32 num_entries (listed above)

VOID* undocumented bu er pointer

UINT32 num_entries (listed above)

DOM_SID2[num_entries] domain SIDs (from Request, listed above).

UINT32 num_

VOID* undocumented domain SID bu er pointer

VOID* undocumented domain name bu er pointer

NAME[num_entries] names to be looked up.

char[] undocumented bytes - falsely translated SID structure?

2.3.12.2 Response

DOM_REF domain reference response

2.4.1.1 Request

VOID* undocumented bu er pointer

UNISTR2 logon server unicode string

UNISTR2 logon client unicode string

char[8] client challenge

2.4.1.2 Response

char[8] server challenge

return 0 - indicates success

2.4.2 LSA Authenticate 2

Note: in between request and response, calculate the client credentials, and check them against the client-calculated credentials (this process uses the previously received client credentials).

Note: neg_ ags in the response is the same as that in the request.

Note: you must take a copy of the client-calculated credentials received here, because they will be used in subsequent authentication packets.

2.4.2.1 Request

LOG_INFO client identi cation info

char[8] client-calculated credentials

UINT8[] padding to 4-byte align with start of SMB header.

Section 2.4. NETLOGON rpc Transact Named Pipe

UINT16 LM20token (same as received in request)

2.5.2 SAM Logon

Note: machine name in response is preceded by two 'n' characters.

Note: NTversion, LMNTtoken, LM20token in response are the same as those given in the request.

Note: user name in the response is presumably the same as that in the request.

2.5.2.1 Request

UINT16 0x0012 - SAM Logon

UINT16 request count

UNISTR machine name

UNISTR user name

STR response mailslot

UINT32 alloweable account

UINT32 domain SID size

char[sid_size] domain SID, of sid_size bytes.

UINT8[] ???? padding to 4?

UINT32 NTwersio2

2.6.1 Net Share Enum

```
C->S Authenticate,Rc
S: Rs = Cred(Ks,Cs), assert(Rc == Cred(Ks,Cc))
S->C Rs
C: assert(Rs == Cred(Ks,Cs))
```

On joining the domain the client will optionally attempt to change its pass-

Section

Part II

Samba Basics

Chapter 4

THE SAMBA DEBUG SYSTEM

4.1 New Output Syntax

The syntax of a debugging log file is represented as:

```
>debugfile< :== { >debugmsg< }

>debugmsg<  :== >debughdr< '\n' >debugtext<

>debughdr<  :== '[' TIME ',' LEVEL ']' FILE ':' [FUNCTION] '(' LINE ')'

>debugtext< :== { >debugline< }

>debugline< :== TEXT '\n'
```

TEXT is a string of characters excluding the newline character.

LEVEL is the DEBUG level of the message (an integer in the range 0..10).

TIME is a timestampharacter.

to send the output to stdout, then you would write

```
DEBUG( 0, ( "This is a %s message.\n", "debug" ) );
```

to send the output to the debug file. All of the normal printf() formatting escapes work.

Note that in the above example the DEBUG message level is set to 0. Mes-

```
  The test returned
[1998/07/30 16:00:51, 0] file.c:function(258)
  True
[1998/07/30 16:00:51, 0] file.c:function(261)
  .
```

Which isn't much use. The format buffer kludge fixes this problem.

4.3 The DEBUGADD() Macro

In addition to the kludgey solution to the broken line problem described

the vslprintf() function, and then passed to format_debug_text(). If you use DEBUGLVL() you will probably print the body of the message using dbgtext().

4.5.2 dbghdr()

Chapter 5

(c)

5.3.14 RSSVAL(buf,pos,val)

sets the value of the unsigned short (16 bit) big-endian integer at o set pos within bu er buf to value val. refered to as "USHORT".

5.3.15 RSIVAL(buf,pos,val)

sets the value of the unsigned 32 bit big-endian integer at o set pos within bu er buf to value val.

5.4 LAN Manager Samba API

7. rparam: a pointer to a pointer which will be set to point to the returned 2.ASCIIZ stringdesupp

2. An adjustment which tells the amount by which pointers in the returned data should be adjusted. This value should be read with SVAL(). Basically, the address of the start of the returned data bu er

Chapter 6

CODING SUGGESTIONS

So you want to add code to Samba ...

One of the daunting tasks facing a programmer attempting to write code for Samba is understanding the various coding conventions used by those most active in the project. These conventions were mostly unwritten and helped

There are lots of platforms that Samba builds on so use caution when adding a call to a library function that is not invoked in existing Samba code. Also note that there are many quite di erent SMB/CIFS clients that Samba tries to support, not all of which follow the SNIA CIFS Technical Reference (or the earlier Microsoft reference documents or the X/Open book on the SMB Standard) perfectly.

Here are some other suggestions:

1. use d_

The suggestions above are simply that, suggestions, but the information may help in reducing the routine rework done on new code. The preceeding list is expected to change routinely as new support routines and macros are added.

Chapter 7

CONTRIBUTING CODE

Here are a few tips and notes that might be useful if you are interested in modifying samba source code and getting it into samba's main branch.

Retrieving the source In order to contribute code to samba, make sure you have the latest source. Retrieving the samba source code from

Chapter 8

MODULES

8.1 Advantages

The new modules system has the following advantages:

Part III

clnt the Client name of the named pipe

srv the Server name of the named pipe

cmds a list of api

Chapter 10

10.1.1 The general interface

A VFS module has three major components:

An initialization function that is called during the module load to register implemented operations.

An operations table representing a mapping between statically de ned module functions and VFS layer operations.

Module functions that do actual work.

```
   struct vfs_handles_pointers {
      ...

      /* File operations */

      struct vfs_handle_struct *open;
      struct vfs_handle_struct *close;
      struct vfs_handle_struct *read;
      struct vfs_handle_struct *write;
      struct vfs_handle_struct *lseek;
      struct vfs_handle_struct *sendfile;

      ...
   } handles;
};
```

This macros SHOULD be used to call any vfs operation. DO NOT ACCESS conn->vfs.ops.* directly !!!

```
...

/* File operations */
#define SMB_VFS_OPEN(conn, fname, flags, mode) \
   ((conn)->vfs.ops.open((conn)->vfs.handles.open,\
    (conn), (fname), (flags), (mode)))
#define SMB_VFS_CLOSE(fsp, fd) \
   ((fsp)->conn->vfs.ops.close(\
   (fsp)->conn->vfs.handles.close, (fsp), (fd)))
#define SMB_VFS_READ(fsp, fd, data, n) \
   ((fsp)->conn->vfs.ops.read(\
   (fsp)->conn->vfs.handles.read,\
    (fsp), (fd), (data), (n)))
#define SMB_VFS_WRITE(fsp, fd, data, n) \
   ((fsp)->conn->vfs.ops.write(\
   (fsp)->conn->vfs.handles.write,\
    (fsp), (fd), (data), (n)))
#define SMB_VFS_LSEEK(fsp, fd, offset, whence) \
   ((fsp)->conn->vfs.ops.lseek(\
```

```
} vfs_op_layer;
```

10.2 The Interaction between the Samba VFS subsystem and the modules

10.2.1 Initialization and registration

As each Samba module a VFS module should have a

```
NTSTATUS vfs_example_init(void);
```

function if it's staticly linked to samba or

```
NTSTATUS init_module(void);
```

function if it's a shared module.

This should be the only non static function inside the module. Global variables should also be static!

The module should register its functions via the

```
NTSTATUS smb_register_vfs(int version, const char *name, vfs_op_tuple *vfs_op_tuples);
```

function.

version should be filled with SMB_VFS_INTERFACE_VERSION

name this is the name witch can be listed in the **vfs objects** parameter to use this module.

vfs_op_tuples this is an array of vfs_op_tuple's. (vfs_op_tuples is descripted in details below.)

For each operation the module wants to provide it has a entry in the vfs_op_tuple array.

```
typedef struct _vfs_op_tuple {
```

```
const char *param;
```

```
#define SMB_VFS_NEXT_SENDFILE(handle, tofd, fsp, fromfd, header, offset, count) \
   ((handle)->vfs_next.ops.sendfile(\
   (handle)->vfs_next.handles.sendfile,\
```

Section

Section

```
    /* get the pointer to our private data
     * return -1 if something failed
     */
    SMB_VFS_HANDLE_GET_DATA(handle, data, struct example_privates, return -1)

    /* do something here...*/
    DEBUG(0,("some_string: %s\n",data->some_string));

    return SMB_VFS_NEXT_CLOSE(handle, fsp, fd);
}
```

11. To make it easy to build 3rd party modules it would be usefull to provide con gure.in, (con gure), install.sh and Make le.in with the module. (Take a look at the example in examples/VFS.)

```
}
```

Overload only the functions you really need to!

10.4.2 Implement OPAQUE functions

If you want to just implement a better version of a default samba opaque function (e.g. like a disk_free() function for a special filesystem) it's ok to just overload that specific function.

If you want to implement a database filesystem or something different from a posix filesystem. Make sure that you overload every vfs operation!!!

Functions your FS does not support should be overloaded by something like this: e.g. for a readonly filesystem.

```
static int example_rename(vfs_handle_struct *handle, connection_struct *conn,
          char *oldname, char *newname)
{
```

Chapter 11

THE SMB.CONF FILE

11.1 Lexical Analysis

Basically, the le is processed on a line by line basis. There are four types of lines that are recognized by the lexical analyzer (params.c):

1. Blank lines - Lines containing only whitespace.

2. Comment lines - Lines beginning with either a semi-colon or a pound sign (';' or '#').

3. Section header lines - Lines beginning with an open square bracket ('[').

4. Parameter lines - Lines beginning with any other character. (The default line type.)

The rst two are handled exclusively by the lexical analyzer, which ignores them. The latter two line types are scanned for

1. - Section names

2. - Parameter names

3. - Parameter values

These are the only tokens passed to the parameter loader (loadparm.c). Parameter names and values are divided from one another by an equal sign: '='.

11.2 Syntax

The syntax of the smb.conf file is as follows:

```
<file>            :==  { <section> } EOF
<section>         :==  <section header> { <parameter line> }
<section header>  :==  '[' NAME ']'
<parameter line>  :==  NAME '=' VALUE NL
```

Basically, this means that

1. a file is made up of zero or more sections, and is terminated by an EOF (we knew that).
2. A section is made up of a section header followed by zero or more parameter lines.
3. A section header is identified by an opening bracket and terminated by the closing bracket. The enclosed NAME identifies the section.
4. A parameter line is divided into a NAME and a VALUE. The *first* equal sign on the line separates the NAME from the VALUE. The VALUE is terminated by a newline character (NL = '*n*n').

Chapter 12

a technical requirement.

```
[global]
   wins server = 192.168.1.2:eth0 192.168.1.3:eth0 192.168.2.2:eth1
```

Using this con guration, nmbd would attempt to register the server's NetBIOS name with one WINS server in each group. Because the "eth0" group has two servers, the second server would only be used when a registration (or resolution) request to the rst server in that group timed out.

NetBIOS name resolution follows.

Debugging and tracing

Chapter 14

TRACING SAMBA SYSTEM CALLS

This le describes how to do a system call trace on Samba to work out what its doing wrong. This is not for the faint of heart, but if you are reading this then you are probably desperate.

Actually its not as bad as the the above makes it sound, just don't expect the output toe5497ibt vrye pettk

doon to bsiness.,Onde ofedainag(es)-417(of)-46(unixs)-417(systesf)-46(isf)-417thatn heyua6(O(y)0ixs)-417(655 0

traceit calsedt1368(di eren)28(t)-765(thungd)-765(on)-765(di eren)27(t)-7

the portabletracervaiilablees051.

Very little happens in a program without a system call so you get lots of

```
[pid 28268] open("/dev/null", O_WRONLY) = -1 EACCES (Permission denied)
```

The process is trying to first open `/dev/null` read-write then read-only. Both fail. This means `/dev/null` has incorrect permissions.

a set of CUPS speci c functions (this is only enabled if the CUPS libraries were located at compile time).

ReplyOpenPrinter

ReplyClosePrinter

RouteRefreshPrinterChangeNotify (RRPCN)

One additional RPC is available to a server, but is never used by the Windows spooler service:

RouteReplyPrinter()

The opnum for all of these RPC's are de ned in include/rpc_spoolss.h

Windows NT print servers use a bizarre method of sending print noti cation event to clients. The process of registering a new change noti cation handle is as follows. The 'C' is for client and the 'S' is for server. All error conditions have been eliminated.

```
C: Obtain handle to printer or to the printer
   server via the standard OpenPrinterEx() call.
S: Respond with a valid handle to object

C: Send a RFFPCN request with the previously obtained
   handle with either (a) set of flags for change events
   to monitor, or (b) a PRINTER_NOTIFY_OPTIONS structure
   containing the event information to monitor.  The windows
   spooler has only been observed to use (b).
S: The <* another missing word*> opens a new TCP session to the client (thus requ
   all print clients to be CIFS servers as well) and sends
   a ReplyOpenPrinter() request to the client.
C: The client responds with a printer handle that can be used to
   send event notification messages.
S: The server replies success to the RFFPCN request.

C: The windows spooler follows the RFFPCN with a RFNPCN
   request to fetch the current values of all monitored
```

request back to the client first. However a request of this nature from the client is often an indication that the previous notification event was not marshalled correctly by the server or a piece of data was wrong.

S: The server closes the internal change notification handle (POLICY_HND) and does not send any further change notification

Chapter 16

NOTES TO PACKAGERS

16.1 Versioning

Please, please update the version number in `source/include/version.h` to include the versioning of your package. This makes it easier to distinguish standard samba builds from custom-build samba builds (distributions often