Network Working Group
INTERNET-DRAFT
draft-leach-cifs-logon-spec-00.txt
Category: Informational
Expires June 3, 1997

Paul J. Leach, Microsoft
Dilip C. Naik, Microsoft

January 3, 1997

CIFS Logon and Pass Through Authentication

Preliminary Draft

STATUS OF THIS MEMO

THIS IS A PRELIMINARY DRAFT OF AN INTERNET-DRAFT. IT DOES NOT REPRESENT THE CONSENSUS OF ANY WORKING GROUP.

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress".

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Distribution of this document is unlimited. Please send comments to the authors or the CIFS mailing list at <cifs@listserv.msn.com>. Discussions of the mailing list are archived at <URL:http://microsoft.ease.lsoft.com/archives/cifs.html>.

ABSTRACT

This specification defines how a certain Common Internet File Systems (CIFS) client accomplishes logging on to a CIFS server. The specification also details how a CIFS server may accomplish pass through authentication.

Table of Contents

Objective

This document details :

- how a CIFS client logs on to a domain
- how a CIFS client logs on to a CIFS server where the CIFS server performs pass through authentication , verifying the client credentials with a Domain Controller.

For convenience, some sections from the CIFS specification have been reproduced in part within this document. Note that the CIFS specification should be considered to be the authoritative reference, in case of any doubts, rather than this document.

1. Prerequisites and suggested reading

- Familiarity with Common Internet File Systems specification (CIFS)
- Familiarity with the CIFS Remote Administration Protocol (RAP) specification.

2. CIFS Domain Logon

CIFS domain logon is a mechanism by which a CIFS server validates a user's credentials with a Domain Controller (DC). For the purposes of this document, a domain is simply a logical grouping of resources such as CIFS servers, user accounts, etc. A Domain Controller is a CIFS server that coordinates synchronization and management activities within a domain. Specific Microsoft products implement the concept of domains with much greater richness and detail, but that is beyond the scope of this document.

A CIFS client logs onto a domain in 3 logical steps:

- Domain Controller Discovery
- Session Setup
- Remote API execution

Each of these are discussed in greater detail in the following sections.

2.1 Domain Controller Discovery

Domain Controller Discovery is the process by means of which a CIFS client locates a Domain Controller (DC). This functionality is implemented using mailslots and special NETBIOS names. Before proceeding to explain the Domain Controller Discovery mechanism, it would be appropriate to discuss NETBIOS names and Mailslots.

Mailslots provide an easy to use mechanism for fast, unreliable unidirectional data transfer. With Microsoft implementations of mailslots, once an application has obtained a handle to a mailslot, the application can write to the mailslot in a fashion very similar to writing to a file. Mailslot writes are implemented using the CIFS Transact SMB which is sent via a datagram to some special Netbios names. Various data structures, which are detailed subsequently within this document, flow as the data portion of the Transact SMB.

2.1.1 NetBIOS Name Notation

NAME(xx) denotes the ASCII string "*NAME*," padded with spaces (0x20) to 15 bytes, with a hex xx value in the 16th byte. For example, the notation "FOOBAR(xx)" indicates a NetBIOS name consisting of the bytes:

```
[69,79,79,65,64,82,20,20,20,20,20,20,20,20,20, xx]
```

String literals that are placeholders and that need to be substituted with their actual values are bracketed within <>. Thus the string <Domain> would become “Redmond” if the domain under consideration is named “Redmond”.
Details of the various NETBIOS names used for browsing are described in Appendix C

2.1.2 Mailslot Protocol Specification

Mailslots provide an easy to use mechanism for fast, unreliable unidirectional data transfer. With Microsoft implementations of mailslots, once an application has obtained a handle to a mailslot, the application can write to the mailslot in a fashion very similar to writing to a file. Mailslot writes are implemented using the CIFS Transact SMB which is sent via a datagram to some special Netbios names. Various data structures, which are detailed subsequently within this document, flow as the data portion of the CIFS Transact SMB.

The only transaction allowed to a mailslot is a mailslot write. Mailslot writes requests are encapsulated in CIFS TRANSACT SMBs. The following table shows the interpretation of the TRANSACT SMB parameters for a mailslot transaction:

Name	Value	Description
Command	SMB_COM_TRANSACTION	
Name	\MAILSLOT\<name>	STRING Name of mail slot to write
SetupCount	3	Always 3 for mailslot writes
Setup[0]	1	Command code == write mailslot
Setup[1]	Ignored	
Setup[2]	Ignored	
TotalDataCount	n	Size of data in bytes to write to the mailslot
Data[n]		The data to write to the mailslot

2.1.3 Primary Domain Controller Location Protocol

This appendix details how a client goes about locating a Primary Domain Controller (PDC). The process is rather involved, because different versions of the Primary Domain Controller have used different versions of the protocol, and hence a client that does not know what protocol is supported by its Primary Domain Controller has to try them all.

A Primary Domain Controller (PDC) for a domain "D" is located by sending a mailslot message containing a NETLOGON_QUERY frame to a NETBIOS name and mailslot "\NET\NETLOGON" and then waiting for a reply mailslot message, which will be sent to the mailslot name specified by the client in the NETLOGON_QUERY structure., and which will contain a NETLOGON_RESPONSE structure. If there is no response after a delay, the message may be retransmitted. The delay MUST be at least twice the expected service time, and the delay should be doubled after each time-out.

If a reply is received, the name of the Primary Domain Controller SHOULD be cached for future use, so as time minimize network traffic. If no reply is received after several retransmissions, the Primary Domain Controller may be declared to be unreachable, and no further attempt to locate it should be made for a while (exactly how long depends on the expected recovery time for a Primary Domain Controller and/or for the network; typically a minute or so, but should be increased after each failure).

The only difference between versions of the protocol is the NETBIOS name to which the message is sent, as follows:

NETBIOS name	name type	PDC's OS version
D(1b)	unique	Windows NT 3.51 or later or compatible
D(1c)	group	Windows NT 3.1 or later or compatible
D(00)	group	all

Clients which are configured to know or are willing to assume what version of the protocol their Primary Domain Controller is running may directly use the appropriate NETBIOS name for that version. Otherwise, they SHOULD first attempt D(1b), since it is unicast and creates the least network traffic; if there is no response, then they SHOULD try the others. They MAY try them in parallel.

The NETLOGON_QUERY structure is defined as :

```
struct NETLOGON_QUERY{
     unsigned char  Opcode;
     char               ComputerName[];
     char               MailslotName[];
     unsigned short     Lm20Token;
} ;
```

Opcode ——Identifies this structure as a NETLOGON_QUERY and has a value of 0x07.

ComputerName ——Specifies the ASCII name of the computer sending the query, and is up to 16 bytes in length. The response is sent to NETBIOS unique name <ComputerName>(00).

MailslotName ——Specifies the ASCII name of the mailslot to which the response is to be sent, and is up to 256 bytes in length; cannot be “\MAILSLOT\LANMAN” or “\MAILSLOT\MSBROWSE” or "\NET\NETLOGON".

Lm20Token - has a value of 0xFFFF.

The response mailslot message contains a NETLOGON_RESPONSE data structure that is defined as the following :

```
struct NETLOGON_RESPONSE
{
     unsigned char  Opcode;
     char               PrimaryDCName[16];
     unsigned short     Lm20Token;
};
```

where

Opcode ——Identifies this structure as a NETLOGON_RESPONSE and has a value of 0x12.

PrimaryDCName ——Specifies the ASCII name of the Primary Domain Controller and is up to 16 bytes in length.

Lm20Token - has a value of 0xFFFF

Note that this procedure to locate a Primary Domain Controller is expensive in terms of network traffic. The Microsoft implementations attempt to alleviate this by caching the PDC Name. Before using the cached PDC Name, a NetServerEnum2 API is remoted to the PDC and a sanity check is performed to ensure that the server type returned indicates a Primary Domain Controller

2.2 Session Setup

The objective of this phase is to validate the client credentials. The CIFS client sends a SessionSetupAndX SMB to the Domain Controller whose identity has just been dicovered, along with a challenge response that is computed as detailed in the CIFS specification. The SessionSetupAndX SMB response indicates whether the Domain Controller was satisfied with the challenge response. The SessiopnSetupAndX SMB response will also indicate if the Domain Controller is dissatisfied with the challenge response, but permits guest access. Obviously, the SessionSetupAndX SMB must be preceded by a Negotiate SMB as detailed in the CIFS specification.

The Negotiate and Session Setup SMBs are detailed in the CIFS document. All of these SMBs are sent to the Domain Controller.

2.3 Remote API execution

Persons unfamiliar with the RAP specification are strongly advised to read the CIFS specification or at least Appendix A at this stage. Sections that follow describe how a CIFS client logs on, logs off and retrieves other significant information such as home directory, etc about a particular user.

2.4 NetwkstaUserLogon

This is a function executed on a remote CIFS server to log on a user. The purpose is to perform checks such as whether the specified user is permitted to logon from the specified computer, whether the specified user is permitted to log on at the given moment, etc. as well as perform housekeeping and statistics updates.

There is a password field in the parameters for this function. However, this field is always set to null before the function is sent on the wire, in order to preserve security. The remote CIFS server ignores this meaningless password that is sent. The remote CIFS server ensures security by checking that the user name and computer name that are in the request parameters are the same used to establish the session and connection to the IPC$ share on the remote CIFS server.

The definition is:

```
unsigned short NetWkstaUserLogon(
    char                *reserved1;
    char                *reserved2;
    unsigned short      sLevel;
    BYTE                bReqBuffer[54];
    unsigned short      cbReqBuffer;
    RCVBUF    pbBuffer;
    RCVBUFLEN           cbBuffer;
    unsigned short      *pcbTotalAvail;
);
```

where:

reserved1 and reserved2 are reserved fields and must be null.

sLevel specifies the level of detail returned. The only legal value is 1.

pbReqBuffer points to the request buffer. This buffer contains parameters that need to be sent to the server. The actual value and structure is defined in the Transaction Request Parameters section.

cbReqBuffer specifies the size, in bytes, of the buffer pointed to by the *pbReqBuffer* parameter. The value must be decimal 54.

pbBuffer points to the buffer to receive the returned data.

cbBuffer specifies the size, in bytes, of the buffer pointed to by the *pbBuffer* parameter.

pcbTotalAvail is a pointer to an unsigned short which gets filled with the total number of data bytes available if the function succeeds.

Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The 16 bit function number for NetWkstaUserLogon which is 132.
- The parameter descriptor string which is "zzWb54WrLh"
- The data descriptor string for the (returned) data which is "WB21BWDWWDDDDDDDzzzD"
- The actual parameters as described by the parameter descriptor string.

The parameters are:

- A null pointer
- Another null pointer
- A 16 bit integer with a value of 1 (corresponding to the "W" in the parameter descriptor string. This represents the level of detail the server is expected to return)
- a byte array of length 54 bytes. These 54 bytes are defined as

```
char wlreq1_name[21];         // User Name
char wlreq1_pad1;             //Pad next field to a word boundary
char wlreq1_password[15];     //Password, set to null, ignored by server
char wlreq1_pad2;             //Pad next field to word boundary
char wlreq1_workstation[16]; //ASCII name of computer
```

- A 16 bit integer with a value of 54
- A 16 bit integer that contains the size of the receive buffer

Transaction Request Data section

There is no data or auxiliary data to send as part of the request.

Transaction Response Parameters section

The transaction response parameters section consists of:

- A 16 bit word indicating the return status. The possible values are:

Code	Value	Description
NERR_Success	0	No errors encountered
ERROR_ACCESS_DENIED	5	User has insufficient privilege
NERR_LogonScriptError	2212	An error occurred while loading or running the logon script
NERR_StandaloneLogon	2214	The logon was not validated by any server
NERR_NonValidatedLogon	2217	The logon server is running an older software version and cannot validate the logon
NERR_InvalidWorkstation	2240	The user is not allowed to logon from this computer

NERR_InvalidLogonHours	2241	The user is not allowed to logon at this time
NERR_PasswordExpired	2242	The user password has expired

- A 16 bit "converter" word.
- A 16 bit number representing the total number of available bytes. This has meaning only if the return status is NERR_Success or ERROR_MORE_DATA. Upon success, this number indicates the number of useful bytes available. Upon failure, this indicates how big the receive buffer needs to be.

Transaction Response Data section

The Transaction response data section contains a data structure user_logon_info_1 which is defined as:

```
struct user_logon_info_1 {
     unsigned short     usrlog1_code;
     char               usrlog1_eff_name[21];
     char               usrlog1_pad_1;
     unsigned short     usrlog1_priv;
     unsigned long      usrlog1_auth_flags;
     unsigned short     usrlog1_num_logons;
     unsigned short     usrlog1_bad_pw_count;
     unsigned long      usrlog1_last_logon;
     unsigned long      usrlog1_last_logoff;
     unsigned long      usrlog1_logoff_time;
     unsigned long      usrlog1_kickoff_time;
     long               usrlog1_password_age;
     unsigned long      usrlog1_pw_can_change;
     unsigned long      usrlog1_pw_must_change;
     char               *usrlog1_computer;
     char               *usrlog1_domain;
     char               *usrlog1_script_path;
     unsigned long      usrlog1_reserved1;
};
```

where:

usrlog1_code specifies the result and can have the following values:

Code	Value	Description
NERR_Success	0	No errors encountered
ERROR_ACCESS_DENIED	5	User has insufficient privilege
NERR_LogonScriptError	2212	An error occurred while loading or running the logon script
NERR_StandaloneLogon	2214	The logon was not validated by any server
NERR_NonValidatedLogon	2217	The logon server is running an older software version and cannot validate the logon
NERR_InvalidWorkstation	2240	The user is not allowed to logon from this computer
NERR_InvalidLogonHours	2241	The user is not allowed to logon at this time
NERR_PasswordExpired	2242	Administrator privilege

usrlog1_eff_name specifies the account to which the user was logged on

usrlog1_pad1 aligns the next data structure element to a word boundary

usrlog1_priv specifies the user's privilege level. The possible values are:

Name	Value	Description
USER_PRIV_GUEST	0	Guest privilege
USER_PRIV_USER	1	User privilege
USER_PRV_ADMIN	2	Administrator privilege

usrlog1_auth_flags specifies the account operator privileges. The possible values are:

Name	Value	Description
AF_OP_PRINT	0	Print operator
AF_OP_COMM	1	Communications operator
AF_OP_SERVER	2	Server operator
AF_OP_ACCOUNTS	3	Accounts operator

usrlog1_num_logons specifies the number of times this user has logged on. A value of -1 means the number of logons is unknown.

usrlog1_bad_pw_count specifies the number of incorrect passwords entered since the last successful logon.

usrlog1_last_logon specifies the time when the user last logged on. This value is stored as the number of seconds elapsed since 00:00:00, January 1, 1970.

usrlog1_last_logoff specifies the time when the user last logged off. This value is stored as the number of seconds elapsed since 00:00:00, January 1, 1970. A value of 0 means the last logoff time is unknown.

usrlog1_logoff_time specifies the time when the user should logoff. This value is stored as the number of seconds elapsed since 00:00:00, Jan 1, 1970. A value of -1 means the user never has to logoff.

usrlog1_kickoff_time specifies the time when the user will be logged off by the system. This value is stored as the number of seconds elapsed since 00:00:00, Jan 1, 1970. A value of -1 means the system will never logoff the user.

usrlog1_password_age specifies the time in seconds since the user last changed his/her password.

usrlog1_password_can_change specifies the time when the user can change the password. This value is stored as the number of seconds elapsed since 00:00:00, Jan 1, 1970. A value of -1 means the user can never change the password.

usrlog1_password_must_change specifies the time when the user must change the password. This value is stored as the number of seconds elapsed since 00:00:00, Jan 1, 1970.

usrlog1_computer specifies the computer where the user is logged on.

usrlog1_script_path specifies the relative path to the user logon script.

usrlog1_reserved is reserved with an undefined value.

The following table defines the valid fields in the user_logon_info_1 structure based upon the return values::

function return code	usrlog1_code element	Valid elements of logoff_info_1
NERR_Success	NERR_Success	All
NERR_Success	NERR_StandaloneLogon	None except usrlog1_code
ERROR_ACCESS_DENIED	NERR_PasswordExpired	None except usrlog1_code
ERROR_ACCESS_DENIED	NERR_InvalidWorkstation	None except usrlog1_code
ERROR_ACCESS_DENIED	NERR_InvalidLogonhours	None except usrlog1_code
ERROR_ACCESS_DENIED	NERR_LogonScriptError	None except usrlog1_code
ERROR_ACCESS_DENIED	ERROR_ACCESS_DENIED	None except usrlog1_code

All other errors	None; the code is meaningless	None

All of the pointers in this data structure need to be treated specially. The pointer is a 32 bit pointer. The higher 16 bits need to be ignored. The converter word returned in the parameters section needs to be subtracted from the lower 16 bits to calculate an offset into the return buffer where this ASCII string resides.

There is no auxiliary data in the response.

2.5 NetwkstaUserLogoff

This is a function executed on a remote CIFS server to log on a user. The purpose is to perform some checks and accomplish housekeeping and statistics updates.

The definition is:

```
unsigned short NetWkstaUserLogoff(
    char            *reserved1;
    char            *reserved2;
    unsigned short  sLevel;
    BYTE            bReqBuffer[54];
    unsigned short  cbReqBuffer;
    REQBUF  pbBuffer;
    REQBUFLEN       cbBuffer;
    unsigned short  *pcbTotalAvail;
);
```

where:

reserved1 and reserved2 are reserved fields and must be null.

sLevel specifies the level of detail returned. The only legal value is 1.

pbReqBuffer points to the request buffer. This buffer contains parameters that need to be sent to the server. The actual value and structure is defined in the Transaction Request Parameters section.

cbReqBuffer specifies the size, in bytes, of the buffer pointed to by the *pbReqBuffer* parameter. The value must be decimal 54.

pbBuffer points to the buffer to receive the returned data.

cbBuffer specifies the size, in bytes, of the buffer pointed to by the *pbBuffer* parameter.

pcbTotalAvail is a pointer to an unsigned short which gets filled with the total number of data bytes available if the function succeeds.

Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The 16 bit function number for NetWkstaUserLogoff which is 133.
- The parameter descriptor string which is "zzWb38WrLh"
- The data descriptor string for the (returned) data which is "WDW"
- The actual parameters as described by the parameter descriptor string.

The parameters are:

- A null pointer

- Another null pointer
- A 16 bit integer with a value of 1 (corresponding to the "W" in the parameter descriptor string. This represents the level of detail the server is expected to return)
- An array of length 38 bytes. These 38 bytes are defined as

```
char            wlreq1_name[21];       // User Name
char            wlreq1_pad1;           //Pad next field to a word boundary
char            wlreq1_workstation[16];    //ASCII name of computer
```

- A 16 bit integer with a value of decimal 38.
- A 16 bit integer that contains the size of the receive buffer

Transaction Request Data section

There is no data or auxiliary data to send as part of the request.

Transaction Response Parameters section

The transaction response parameters section consists of:

- A 16 bit word indicating the return status. The possible values are:

Code	Value	Description
NERR_Success	0	No errors encountered
NERR_StandaloneLogon	2214	The logon was not validated by any server
NERR_NonValidatedLogon	2217	The logon server is running an older software version and cannot validate the logoff

- A 16 bit "converter" word.
- A 16 bit number representing the total number of available bytes. This has meaning only if the return status is NERR_Success or ERROR_MORE_DATA. Upon success, this number indicates the number of useful bytes available. Upon failure, this indicates how big the receive buffer needs to be.

Transaction Response Data section

The Transaction response data section contains a data structure user_logoff_info_1 which is defined as:

```
struct user_logoff_info_1 {
        unsigned short  usrlogf1_code;
        unsigned long   usrlogf1_duration;
        unsigned short  usrlogf1_num_logons;
};
```

where:

usrlogf1_code specifies the result and can have the following values:

Code	Value	Description
NERR_Success	0	No errors encountered
ERROR_ACCESS_DENIED	5	User has insufficient privilege
NERR_InvalidWorkstation	2240	The user is not allowed to logon from this computer

usrlogf1_duration specifies the time in number of seconds for which the user was logged

usrlogf1_num_logons specifies the number of times this user has logged on. A value of -1 indicates the number is unknown.

The following table defines the valid fields in the logoff_info_1 structure based upon the return values::

function return code	usrlogf11_code element	Valid elements of logoff_info_1
NERR_Success	NERR_Success	All
NERR_Success	NERR_StandaloneLogon	None except usrlogf1_code
All other errors	None; the code is meaningless	None

There is no auxiliary data in the response.

2.6 NetUserGetInfo

This is a function executed on a remote CIFS server to obtain detailed information about a particular user.

The definition is:

```
unsigned short NetUserGetInfo(
    char                *pszUser;
    unsigned short      sLevel;
    RCVBUF   pBuffer;
    RCVBUFLEN           cbBuffer;
    unsigned short      *pcbTotalAvail;
);
```

where:

pszUser points to a null terminated ASCII string signifying the name of the user for which information should be retrieved.

sLevel specifies the level of detail returned. The only legal value is 11.

pbBuffer points to the buffer to receive the returned data.

cbBuffer specifies the size, in bytes, of the buffer pointed to by the *pbBuffer* parameter.

pcbTotalAvail is a pointer to an unsigned short which gets filled with the total number of data bytes available if the function succeeds.

Transaction Request Parameters section

The Transaction request parameters section in this instance contains:

- The 16 bit function number for NetUserGetInfo which is 56.
- The parameter descriptor string which is "zWrLh"
- The data descriptor string for the (returned) data which is "B21BzzzWDDzzDDWWzWzDWb21W"
- The actual parameters as described by the parameter descriptor string.

The parameters are:

- A null terminated ASCII string indicating the user for which information should be retrieved.
- A 16 bit integer with a value of decimal 11 (corresponding to the "W" in the parameter descriptor string. This represents the level of detail the server is expected to return)
- A 16 bit integer that contains the size of the receive buffer

Transaction Request Data section

There is no data or auxiliary data to send as part of the request.

Transaction Response Parameters section

The transaction response parameters section consists of:

- A 16 bit word indicating the return status. The possible values are:

Code	Value	Description
NERR_Success	0	No errors encountered
ERROR_ACCESS_DENIED	5	User has insufficient privilege
ERROR_MORE_DATA	234	additional data is available
NERR_BufTooSmall	2123	The supplied buffer is too small
NERR_UserNotFound	2221	The user name was not found

- A 16 bit "converter" word.
- A 16 bit number representing the total number of available bytes. This has meaning only if the return status is NERR_Success or ERROR_MORE_DATA. Upon success, this number indicates the number of useful bytes available. Upon failure, this indicates how big the receive buffer needs to be.

Transaction Response Data section

The Transaction response data section contains a data structure user_logon_info_1 which is defined as:

```
struct user_info_11 {
     char               usri11_name[21];
     char               usri11_pad;
     char               *usri11_comment;
     char               *usri11_usr_comment;
     unsigned short     usri11_priv;
     unsigned long      usri11_auth_flags;
     long               usri11_password_age;
     char               *usri11_homedir;
     char               *usri11_parms;
     long               usri11_last_logon;
     long               usri11_last_logoff;
     unsigned short     usri11_bad_pw_count;
     unsigned short     usri11_num_logons;
     char               *usri11_logon_server;
     unsigned short     usri11_country_code;
     char               *usri11_workstations;
     unsigned long      usri11_max_storage;
     unsigned short     usri11_units_per_week;
     unsigned char      *usri11_logon_hours;
     unsigned short     usri11_code_page;
};
```

where:

usri11_name specifies the user name for which information is retireved

usri11_pad aligns the next data structure element to a word boundary

usri11_comment is a null terminated ASCII comment

usri11_user_comment is a null terminated ASCII comment about the user

usri11_priv specifies the level of the privilege assigned to the user. The possible values are:

Name	Value	Description
USER_PRIV_GUEST	0	Guest privilege
USER_PRIV_USER	1	User privilege
USER_PRV_ADMIN	2	Administrator privilege

usri11_auth_flags specifies the account operator privileges. The possible values are:

Name	Value	Description
AF_OP_PRINT	0	Print operator
AF_OP_COMM	1	Communications operator
AF_OP_SERVER	2	Server operator
AF_OP_ACCOUNTS	3	Accounts operator

usri11_password_age specifies how many seconds have elapsed since the password was last changed.

usri11_home_dir points to a null terminated ASCII string that contains the path name of the user's home directory.

usri11_parms points to a null terminated ASCII string that is set aside for use by applications.

usri11_last_logon specifies the time when the user last logged on. This value is stored as the number of seconds elapsed since 00:00:00, January 1, 1970.

usri11_last_logoff specifies the time when the user last logged off. This value is stored as the number of seconds elapsed since 00:00:00, January 1, 1970. A value of 0 means the last logoff time is unknown.

usri11_bad_pw_count specifies the number of incorrect passwords entered since the last successful logon.

usri11_log1_num_logons specifies the number of times this user has logged on. A value of -1 means the number of logons is unknown.

usri11_logon_server points to a null terminated ASCII string that contains the name of the server to which logon requests are sent. A null string indicates logon requests should be sent to the domain controller.

usri11_country_code specifies the country code for the user's language of choice.

usri11_workstations points to a null terminated ASCII string that contains the names of workstations the user may log on from. There may be up to 8 workstations, with the names separated by commas. A null strings indicates there are no restrictions.

usri11_max_storage specifies the maximum amount of disk space the user can occupy. A value of 0xffffffff indicates there are no restrictions.

usri11_units_per_week specifies the equal number of time units into which a week is divided. This value must be equal to 168.

usri11_logon_hours points to a 21 byte (168 bits) string that specifies the time during which the user can log on. Each bit represents one unique hour in a week. The first bit (bit 0, word 0) is Sunday, 0:00 to 0:59, the second bit (bit 1, word 0) is Sunday, 1:00 to 1:59 and so on. A null pointer indicates there are no restrictions.

usri11_code_page specifies the code page for the user's language of choice

All of the pointers in this data structure need to be treated specially. The pointer is a 32 bit pointer. The higher 16 bits need to be ignored. The converter word returned in the parameters section needs to be subtracted from the lower 16 bits to calculate an offset into the return buffer where this ASCII string resides.

There is no auxiliary data in the response.

3. CIFS pass through authentication

CIFS pass through authentication is a mechanism employed by a CIFS server to validate user credentials with a Domain Controller and the grant the user access to a resource on the CIFS server, based upon successful validation of the user credentials by the Domain Controller.

Note that a CIFS server can do pass through authentication to only a single domain. Thus the name of the domain is essentially known before a user even attempts to connect to the CIFS server. The CIFS server locates the Domain Controller of this single domain of interest using the mechanism described in section 3.1.3. This mechanism is expensive in terms of network traffic, so the CIFS server caches the name of the Domain Controller. The CIFS server can verify this cached information by sending a NetServerEnum2 RAP request to the Domain Controller and checking that the returned information still indicates the server to be a Domain Controller. Complete details of the RAP specification as well as details of the NetServerEnum2 RAP request may be found in the CIFS Remote Administration Protocol specification.

Consider the case of a CIFS server running with user level security. The CIFS document describes a user level (security) server as a server that requires clients to provide a user name and corresponding password to connect to any resources shared by the server.

Consider a CIFS client seeking to connect to this CIFS server. The client is prepared to submit it's credentials (user name and challenge response). The CIFS server does not have an accounts database that can establish the validity of the user credentials. This is the situation in which a CIFS server resorts to pass through authentication. The steps involved in pass through authentication are:

- The CIFS client sends a negotiate SMB to the CIFS server
- The CIFS server verifies the cached Domain Controller name (as described above)
- If the cached name is invalid, the CIFS server does a Domain Controller Discovery
- The CIFS server sends a NEGOTIATE SMB to the Domain Controller
- The NEGOTIATE response along with the challenge is saved by the CIFS server
- The CIFS server sends a NEGOTIATE response (to client) using the saved challenge
- The CIFS client computes the challenge response as detailed in the CIFS specification, and then challenge response is sent as part of a SessionSetupAndX SMB
- The CIFS server extracts the challenge response from above SMB
- The CIFS server sends it's own SessionSetupAndX SMB to the domain controller using the extracted challenge response
- The Domain Controller sends a SessionSetupAndX response to the CIFS server. This response will be successful if the CIFS client had provided the correct response.
- The CIFS server tears down the session with the Domain Controller that was established using user credentials. This is accomplished by means of a LogOffAndX SMB.
- The CIFS server sends a SessionSetupAndX response to the CIFS client. This response is based upon the response from the Domain Controller.

4. Appendix A - Remote Administration Protocol

A RAP service request is sent to the server encapsulated in a Transact2 request SMB and the server sends back a Transact2 SMB response. An attribute of the Transact2 SMB is that it divides the payload of request and response messages into two sections: a *parameters* section and a *data* section. As might be expected from the nomenclature, RAP service parameters are sent in the parameters section of a Transact2 SMB, and the data buffer in the data section. Therefore, to define a service protocol, it is necessary to define the formats of the parameter and data sections of the Transact2 request and response.

This is done in two stages. First, a C-like declaration notation is used to define descriptor strings, and then the descriptor strings define the formats of the parameter and data sections.. Note well: even though the declarations may look like a programming interface, they are not: they are a notation for describing the contents of RAP requests and responses; an implementation on any particular system can use any programming interface to RAP services that is appropriate to that system.

4.1 Notation

Parameter descriptor strings are defined using a C-like function declaration; data descriptor and auxiliary data descriptor strings are defined using a C-like structure declaration.

Parameter descriptor strings are defined with the following C-like function declaration syntax:

```
rap-service         = "unsigned short" service-name "(" parameters ");"
service-name        = <upper and lower case alpha and numeric>
```

The return type of the function is always "unsigned short", and represents the status code from the function. The service-name is for documentation purposes.

```
parameters          = parameter [ ";" parameter ]
```

The parameter descriptor string for the service is the concatenation of the descriptor characters for the parameters.

```
parameter           = [ "const" ] param-data-type parameter-name
                      [ "[" size "]" ]
param-data-type     = <from parameter descriptor tables below>
parameter-name      = <upper and lower case alpha and numeric>
size                = <string of ASCII 0-9>
```

The descriptor character for a parameter is determined by looking up the data-type in the tables below for request or response parameter descriptors. The parameter-name is for documentation purposes. If there is a size following the parameter-name, then it is placed in the descriptor string following the descriptor character.

Data and auxiliary data descriptor strings are defined with the following C-like structure declaration syntax:

```
rap-struct          = "struct" struct-name "{" members "}"
```

The descriptor string for the struct is the concatenation of the descriptor characters for the members. The struct-name is for documentation purposes.

```
members             = member [  ";" member ]
member              = member-data-type member-name [ "[" size "]" ]
member-data-type    = <from data descriptor tables below>
```

The descriptor character for a member is determined by looking up the data-type in the tables below for data descriptors. The member-name is for documentation purposes. If there is a size following the member-name, then it is placed in the descriptor string following the descriptor character.

4.2 Descriptors

The following section contain tables that specify the descriptor character and the notation for each data type for that data type.

4.2.1 Request Parameter Descriptors

Descriptor	Data Type	Format
W	**unsigned short**	indicates parameter type of 16 bit integer (word).
D	**unsigned long**	indicates parameter type of 32 bit integer (dword).
b	**BYTE**	indicates bytes (octets). May be followed by an ASCII number indicating number of bytes..
O	**NULL**	indicates a NULL pointer
z	**char**	indicates a NULL terminated ASCII string present in the parameter area
F	**PAD**	indicates Pad bytes (octets). May be followed by an ASCII number indicating the number of bytes
r	**RCVBUF**	pointer to receive data buffer in response parameter section
L	**RCVBUFLEN**	16 bit integer containing length of receive data buffer in (16 bit) words
s	**SNDBUF**	pointer to send data buffer in request parameter section

T	**SNDBUFLEN**	16 bit integer containing length of send data buffer in words

4.2.2 Response Parameter Descriptors

Descriptor	Data Type	Format
==========	=========	=====
g	**BYTE ***	indicates a byte is to be received. May be followed by an ASCII number indicating number of bytes to receive
h	**unsigned short ***	indicates a word is to be received
i	**unsigned long ***	indicates a dword is to be received
e	**ENTCOUNT**	indicates a word is to be received which indicates the number of entries returned

4.2.3 Data Descriptors

Descriptor	Data Type	Format
==========	=========	=====
W	**unsigned short**	indicates data type of 16 bit integer (word). Descriptor char may be followed by an ASCII number indicating the number of 16 bit words present
D	**unsigned long**	indicates data type of 32 bit integer (dword). Descriptor char may be followed by an ASCII number indicating the number of 32 bit words present
B	**BYTE**	indicates item of data type 8 bit byte (octet). The indicated number of bytes are present in the data. Descriptor char may be followed by an ASCII number indicating the number of 8 bit bytes present
O	**NULL**	indicates a NULL pointer
z	**char ***	indicates a 32 bit pointer to a NULL terminated ASCII string is present in the response parameter area. The actual string is in the response data area and the pointer in the parameter area points to the string in the data area. The high word of the pointer should be ignored. The converter word present in the response parameter section should be subtracted from the low 16 bit value to obtain an offset into the data area indicating where the data area resides.
N	**AUXCOUNT**	indicates number of auxiliary data structures. The transaction response data section contains an unsigned 16 bit number corresponding to this data item.

4.3 Transaction Request Parameters section

The parameters and data being sent and received are described by ASCII descriptor strings. These descriptor strings are described in section 4.2.

The parameters section of the Transact2 SMB request contains the following (in the order described)

- The function number: an unsigned short 16 bit integer identifying the function being remoted
- The parameter descriptor string: a null terminated ASCII string
- The data descriptor string: a null terminated ASCII string.
- The request parameters, as described by the parameter descriptor string, in the order that the request parameter descriptor characters appear in the parameter descriptor string
- An optional auxiliary data descriptor string: a null terminated ASCII string. It will be present if there is an auxiliary data structure count in the primary data struct (an "N" descriptor in the data descriptor string).

RAP requires that the length of the return parameters be less than or equal to the length of the parameters being sent; this requirement is made to simply buffer management in implementations. This is reasonable as the functions were designed to return

data in the data section and use the return parameters for items like data length, handles, etc. If need be, this restriction can be circumvented by filling in some pad bytes into the parameters being sent.

4.4 Transaction Request Data section

The Data section for the transaction request is present if the parameter description string contains an "s" (SENDBUF) descriptor. If present, it contains:

- A primary data struct, as described by the data descriptor string
- Zero or more instances of the auxiliary data struct, as described by the auxiliary data descriptor string. The number of instances is determined by the value of the an auxiliary data structure count member of the primary data struct, indicated by the "N" (AUXCOUNT) descriptor. The auxiliary data is present only if the auxiliary data descriptor string is non null.
- Possibly some pad bytes
- The heap: the data referenced by pointers in the primary and auxiliary data structs.

4.5 Transaction Response Parameters section

The response sent by the server contains a parameter section which consists of:

- A 16 bit integer indicating the status or return code. The possible values for different functions are different.
- A 16 bit converter word, used adjust pointers to information in the response data section. Pointers returned within the response buffer are 32 bit pointers. The high order 16 bit word should be ignored. The converter word needs to be subtracted from the low order 16 bit word to arrive at an offset into the response buffer.
- The response parameters, as described by the parameter descriptor string, in the order that the response parameter descriptor characters appear in the parameter descriptor string.

4.6 Transaction Response Data section

The Data section for the transaction response is present if the parameter description string contains an "r" (RCVBUF) descriptor. If present, it contains:

- Zero or more entries. The number of entries is determined by the value of the entry count parameter, indicated by the "e"(ENTCOUNT) descriptor. Each entry contains:
    - A primary data struct, as described by the data descriptor string
    - Zero or more instances of the auxiliary data struct, as described by the auxiliary data descriptor string. The number of instances is determined by the value of the AUXCOUNT member of the primary data struct (whose descriptor is "N"). The auxiliary data is present only if the auxiliary data descriptor string is non null.
- Possibly some pad bytes
- The heap: the data referenced by pointers in the primary and auxiliary data structs.

5. Author's Addresses

Paul Leach
Dilip Naik
Microsoft
1 Microsoft Way
Redmond, WA 98052
paulle@microsoft.com
v-dilipn@microsoft.com