CIFS Authentication Protocols Specification

Paul J. Leach

Microsoft

Preliminary Draft – do not cite

Author's draft: 4

This is a preliminary draft of a specification of a proposed new version of the CIFS authentication protocol. It is supplied here as a standalone document for ease of review; if accepted and implemented, it may be incorporated into a future release of the CIFS specification. (This specification may change without notice, and should not be construed as a product commitment from Microsoft Corporation.)

There are two sections: one about the authentication protocols themselves; and one about how the protocol to use is negotiated and the protocol messages transported in CIFS requests and responses.

1. CIFS Authentication Protocols

This section defines the CIFS session and message authentication protocols. Session authentication is done via a challenge response protocol based upon the shared knowledge of the user's password. Message authentication is done by attaching a message authentication code (MAC) to each message.

1.1 Overview

There are several variations on one basic session authentication scheme: To gain authenticated access to server resources, the server sends a "challenge" to the client, which the client responds to in a way that proves it knows the client's password: a "response" is created from the challenge by encrypting it (and possibly a nonce of the client's choice) with a 168 bit "session key" computed from the user's password. The response, or a subset of it, and client nonce are then returned to the server, which can validate the response by performing the same computation. The client and server negotiate which variation to use by a mechanism described in the next section (being careful to not permit a downgrade attack).

We describe the session authentication protocol as if the CIFS server keeps a client's password, but an implementation might actually store the passwords on a key distribution server (KDS) and have servers use a protocol outside the scope of this specification to enable them to perform the steps required by this protocol.

Once the session is authenticated, subsequent messages may be authenticated by computing a MAC for each message and attaching it to the message. The MAC used is a keyed MD5 construction similar to that used in IPSec [RFC 1828], using a "MAC key" computed from the session key, and the response to the server's challenge. The MAC is over both the message text and an implicit sequence number, to prevent replay.

1.2 Definitions

Let

`E(K, D)`

denote the DES block mode encryption function [FIPS 81] , which accepts a seven byte key (K) and an eight byte data block (D) and produces an eight byte encrypted data block as its value.

`Ex(K,D)`

denote the extension of DES to longer keys and data blocks. If the data to be encrypted is longer than eight bytes, the encryption function is applied to each block of eight bytes in sequence and the results are concatenated together. If the key is longer than seven bytes, each 8 byte block of data is first completely encrypted using the first seven bytes of the key, then the second seven bytes, etc., appending the results each time. For example, to encrypt the 16 byte quantity D0D1 with the 14 byte key K0K1,

```
Ex(K0K1,D0D1) = concat(E(K0,D0),E(K0,D1),E(K1,D0),E(K1,D1))
```

`concat(A, B, …, Z)`
`is the result of concatenating the byte strings A, B, … Z`

`head(S, N)`
denote the first N bytes of the byte string S.

`swab(S)`
denote the byte string obtained by reversing the order of the bits in each byte of S, i.e., if S is byte string of length one, with the value 0x37 then swab(S) is 0xEC.

`zeros(N)`
denote a byte string of length N whose bytes all have value 0 (zero).

`ones(N)`
denote a byte string of length N whose bytes all have value 255.

`xor(A, B)`
denote a byte string formed by the bytewise logical "xor" of each of the bytes in A and B.

`and(A, B)`
denote a byte string formed by the bytewise logical "and" of each of the bytes in A and B.

`substr(S, A, B)`
denote a byte string of length N obtained by taking N bytes of S starting at byte A. The first byte is numbered zero. I.e., if S is the string "NONCE" then substr(S, 0, 2) is "NO".

1.3 Session Keys

The session authentication protocol is the same for all dialects, but the computation of the session keys is different depending on the dialect and the authentication negotiation described below.

1.3.1 NT Session Key

The session key S21 and partial MAC key S16 are computed as

```
S16 = MD4(U(PN))
S21 = concat(S16, zeros(5))
```

where

- PN is a string containing the user's password in clear text, case sensitive, no maximum length
- U(x) of an ASCII string "x" is that string converted to Unicode
- MD4(x) of an byte string "x" is the 16 byte MD4 message digest [RFC 1320] of that string

1.3.2 LM Session Key

The session key S21 and partial MAC key S16 are computed as

```
S16X = Ex(swab(P14),N8)
S21 = concat(S16X, zeros(5))
S16 = concat(head(S16X, 8), zeros(8))
```

where

- o P14 is a 14 byte string containing the user's password in clear text, upper cased, padded with spaces
- o N8 is an 8 byte string whose value is available from Microsoft upon request.

1.4 Session Authentication Protocol

The session authentication protocol has the following steps:

- o The server chooses a unique 8 byte challenge C8 (a "nonce" that has never been used before and will not be reused) and sends it to the client
- o The client computes

```
RN = Ex(S21,C8)
```

- o The client sends the 24 byte response RN to the server
- o The server computes RN as above and compares the received response with its computed value for RN; if equal, the client has authenticated.

1.6 Message authentication code

The MAC is the keyed MD5 construction:

```
MAC(K, text) = head(MD5(concat(K, text)), 8)
```

Where:

`MD5`
is the MD5 hash function; see RFC 1321

`K`
is the key

`text`
is the message whose MAC is being computed

See RFC 1828 for an example of Keyed-MD5 applied to IP security. If not used properly, keyed MD5 may have weaknesses as a MAC. Iterative hashes such as MD5 may be subject to message extension attacks and to cryptanalysis [Kal 95]. This construction is not subject to the problems identified there, because the text contains an explicit length, which prevents message extension attacks; and because there are always two iterations of the compression function, and only 64 bits of the hash are output, which prevents known cryptanalysis techniques.

1.7 MAC key

The MAC key is computed as follows:

```
K = concat(S16, RN)
```

Where S16 and RN are as above. K is either 40 or 44 bytes long, depending on the length of RN.

1.8 Message Authentication Protocol

Once the session has been authenticated, each message can be authenticated as well. This will prevent MITM attacks, replay attacks, and active message modification attacks. (See Security Considerations, below.)

Let

SN
: be a request sequence number, initially set to 0. Both client and server have one SN for each connection between them.

RSN
: is the sequence number expected on the response to a request.

req_msg
: be a request message

rsp_msg
: be a response message

The SN is logically contained in each message, as defined below in the section on authenticated message transport, and participates in the computation of the MAC, as defined in that section.

Then for or each message sent in the session the following procedure is followed:

- Client computes MAC(req_msg) using SN, and sends it to the server in the request message. If there are multiple messages in the request, each uses the same SN. If there are multiple requests in the message (using the "AndX" facility) , then the MAC is calculated as if it were a single large request.
- Client increments its SN and saves it as RSN
- Client increments its SN – this is the SN it will use in its next request
- Server receives each req_msg, validates MAC(req_msg) using SN, and responds ACCESS_DENIED if invalid
- Server increments its SN and saves it as RSN
- Server increments its SN – this is the SN it will expect in the next request
- For each response message for this request, computes MAC(rsp_msg) using RSN, and sends it to client in the response message. If there are multiple responses in the message (using the "AndX" facility) , then the MAC is calculated as if it were a single large response.
- Client receives each rsp_msg, validates MAC(rsp_msg) using RSN, and discards the response message if invalid

2. Security Level and Authentication Protocol Negotiation

The `SMB_COM_NEGPROT` response from a server has the following bits in its *SecurityMode* field:

```
#define NEGOTIATE_SECURITY_USER_LEVEL               0x01
#define NEGOTIATE_SECURITY_CHALLENGE_RESPONSE       0x02
#define NEGOTIATE_SECURITY_SIGNATURES_ENABLED       0x04
#define NEGOTIATE_SECURITY_SIGNATURES_REQUIRED      0x08
```

If NEGOTIATE_SECURITY_USER_LEVEL is set, then "user level" security is in effect for all the shares on the server. This means that the client MUST establish a session (with SMB_COM_SESSION_SETUP_ANX) to authenticate the user before connecting to a share, and the password to use in the authentication protocol described above is the user's password. If NEGOTIATE_SECURITY_USER_LEVEL is clear, then "share level" security is in effect for all the shares in the server. This means that a session need not be established, and the password to use in the authentication protocol is a password for the share.

If NEGOTIATE_SECURITY_CHALLENGE_RESPONSE is clear, then the server is requesting plaintext passwords. Clients MUST be able to be configured to refuse requests for plaintext passwords (in order to prevent downgrade attacks by rogue or spoofing servers); this SHOULD be the default. Servers MUST be able to be configured to refuse plaintext passwords (in order to make password guessing attacks harder).

If NEGOTIATE_SECURITY_CHALLENGE_RESPONSE is set, then the server supports the challenge/response session authentication protocol described above, and clients SHOULD use it. Servers MAY refuse connections that do not use it, and MUST be able to be configured to do so.

If the dialect is earlier than "NTLM 0.12" then the client computes the response using the "LM session key". If the dialect is "NTLM 0.12" then the client MAY compute the response either using the "LM session key", or the "NT session key", or both. The server MAY choose to refuse responses computed using the "LM session key", and MUST be able to be configured to do so (in order to protect against downgrade attacks).

If NEGOTIATE_SECURITY_SIGNATURES_ENABLED is set, then the server supports the message authentication protocol described above, and the client MAY use it. This bit may only be set if NEGOTIATE_SECURITY_CHALLENGE_RESPONSE is set. Clients that support security signatures MUST be configurable to refuse to connect to servers that do not have NEGOTIATE_SECURITY_SIGNATURES_ENABLED set. Clients that are so configured MUST refuse to connect to servers that have NEGOTIATE_SECURITY_CHALLENGE_RESPONSE set and do not have NEGOTIATE_SECURITY_USER_LEVEL set (i.e., they must not use challenge/response authentication to a server using "share-level" security).

If NEGOTIATE_SECURITY_SIGNATURES_REQUIRED is set, then the server requires the use of the message authentication protocol described above, and the client MUST use it. This bit may only be set if NEGOTIATE_SECURITY_SIGNATURES_ENABLED is set. This bit MUST NOT be set. if NEGOTIATE_SECURITY_USER_LEVEL is clear (i.e., for servers using "share level" security).

The SMB_COM_SESSION_SETUP_ANDX request has the following bit in the *Flags2* field:

```
#define SMB_FLAGS2_SMB_SECURITY_SIGNATURE 0x0004
```

To use message authentication, the client sets `SMB_FLAGS2_SMB_SECURITY_SIGNATURE` in a SMB_COM_SESSION_SETUP_ANDX request to the server, and includes a MAC calculated as described above. If the resulting session is non-null and non-guest, then the SMB_COM_SESSION_SETUP_ANDX response and all subsequent SMB requests and responses MUST include a MAC calculated as described above. The first non-null, non-guest session determines the key to be used for the MAC for all subsequent sessions.

Message authentication may only be requested when the "NTML 0.12" dialect has been negotiated. If message authentication is used, raw mode MUST not be used (because some raw mode messages have no headers in which to carry the MAC).

No matter which authentication protocol is negotiated, servers MUST be able to be configured to "lock out" accounts that make too many failed authentication attempts, in order to foil brute force attacks on the protocol.

2.1 Plaintext password transport

If plaintext password authentication was negotiated, clients send the plaintext password in the first `SMB_COM_TREE_CONNECT`, `SMB_COM_TREE_CONNECT_ANDX`, and/or `SMB_COM_SESSION_SETUP_ANDX` request which follows the `SMB_COM_NEGPROT` message exchange. The SMB field used to contain the response depends upon the request:

- *Password* in `SMB_COM_TREE_CONNECT`
- *Password* in `SMB_COM_TREE_CONNECT_ANDX`
- *AccountPassword* `in SMB_COM_SESSION_SETUP_ANDX` in dialects prior to "NTLM 0.12"
- *CaseInsensitivePassword* in `SMB_COM_SESSION_SETUP_ANDX` in the "NTLM 0.12" dialect
- *CaseSensitivePassword* in `SMB_COM_SESSION_SETUP_ANDX` in the "NTLM 0.12" dialect

2.2 Challenge/response transport

The challenge C8 from the server to the client is contained in the *EncryptionKey* field in the `SMB_COM_NEGPROT` response. (Note: the name "*EncryptionKey*" is historical -- it doesn't actually hold an encryption key.)

Clients send the cleartext password, or the response to the challenge, in the first `SMB_COM_TREE_CONNECT`, `SMB_COM_TREE_CONNECT_ANDX`, and/or `SMB_COM_SESSION_SETUP_ANDX` request which follows the `SMB_COM_NEGPROT` message exchange. The SMB field used to contain the response depends upon the request:

- *Password* in `SMB_COM_TREE_CONNECT`
- *Password* in `SMB_COM_TREE_CONNECT_ANDX`
- *AccountPassword* `in SMB_COM_SESSION_SETUP_ANDX` in dialects prior to "NTLM 0.12"
- *CaseInsensitivePassword* in `SMB_COM_SESSION_SETUP_ANDX` for a response computed using the "LM session key" in the "NTLM 0.12" dialect
- *CaseSensitivePassword* in `SMB_COM_SESSION_SETUP_ANDX` for a response computed using the "NT session key" in the "NTLM 0.12" dialect

(Note: again, the names are historical, and do not reflect this usage.)

2.3 MAC transport

In each message that contains a MAC, the following bit is set in the flags2 field:

```
#define SMB_FLAGS2_SMB_SECURITY_SIGNATURES 0x0004
```

The SMB header has been modified to include a place for the MAC:

```
typedef struct _SMB_HEADER {
```

```
    UCHAR Protocol[4];              // Contains 0xFF,'SMB'
    UCHAR Command;                  // Command code
    UCHAR ErrorClass;               // Error class
    UCHAR Reserved;                 // Reserved for future use
    USHORT Error ;                  // Error code
    UCHAR Flags;                    // Flags
    USHORT Flags2 ;                 // More flags
    union {
        USHORT Reserved2[6];        // Reserved for future use
        struct {
            USHORT PidHigh;         // High part of PID
                  // Client must send the correct Signature
                  // for this SMB to be accepted.
            UCHAR SecuritySignature[8];
        };
    };
   ....
} SMB_HEADER;
```

The sender of a message inserts the sequence number SSN into the message by putting it into the first 4 bytes of the SecuritySignature field and zeroing the last 4 bytes, computes the MAC over the entire message, then puts the MAC in the field. The receiver of a message validates the MAC by extracting the value of the SecuritySignature field, putting its ESN into the first 4 bytes of the SecuritySignature field and zeroing the last 4 bytes, computing the MAC, and comparing it to the extracted value.

Oplock break messages from the server to the client may not use message authentication, even if it has been negotiated.

3. Security considerations

(These are already in the main CIFS spec, of which this will eventually become a part.) Relevant additions:

Clients that require the use of message authentication will be protected against man-in-the-middle attacks and downgrade attacks by rogue or counterfeit servers. Servers that require the use of message authentication will be protected against man-in-the-middle attacks and downgrade attacks by rogue clients.

4. References

[FIPS 81] DES, FIPS PUB xxx

[RFC 1320] RFC 1320, R. Rivest, The MD4 Message-Digest Algorithm

[RFC 1321] RFC 1321, R. Rivest, The MD5 Message-Digest Algorithm

[RFC 1828] RFC 1828, P. Metzger, W. Simpson, "IP Authentication using Keyed MD5", August 1995

{Kal 95] B. Kaliski, M.Robshaw, "Message Authentication with MD5", CryptoBytes, Sping 1995, RSA Inc, (http://www.rsa.com/rsalabs/pubs/cryptobytes/spring95/md5.htm)