opengroup.org

NTLM

23–28 Minuten

---

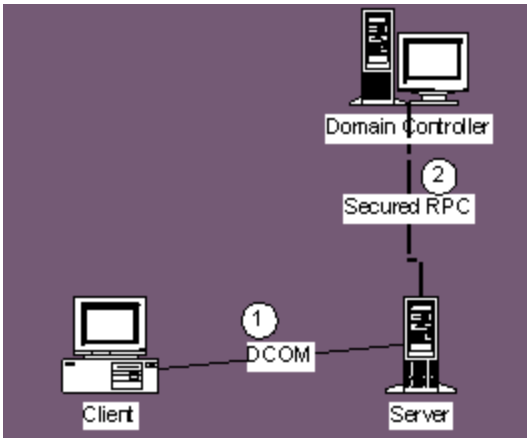
11 NTLM

This document describes and defines the DCOM NTLM Passthrough Security system. It not only describes the passthrough architecture and mechanism, but also the NTLM security implementation.

11.1 NTLM Passthrough Architecture

This section describes the high level architecture of the passthrough security system. The passthrough security system consists of three actors: client, server, and domain controller. The client initiates the DCOM conversation. The server hosts the NT LAN Manager (LM) passthrough security system. And a Windows NT domain controller performs the authentication. Between these three actors there are two communication channels, DCOM and Secured RPC. The connection between the client and server (1) uses DCOM. The connection between the server and domain controller (2) uses secured RPC.

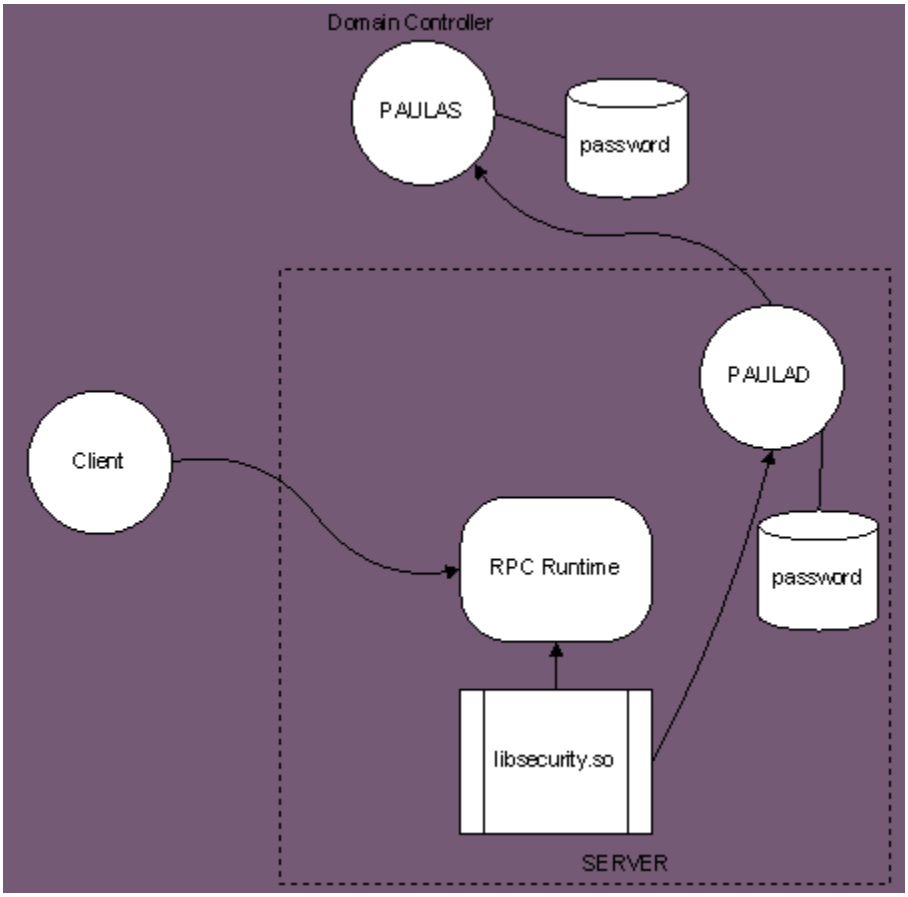
Figure 11-1: NTLM Use of DCOM and Secured RPC



When the client communicates with the server, an authentication handshake takes place. NTLM security specifies a challenge/ response protocol that must be followed in order to authenticate the client. Since the server does not contain the Windows NT security system, it forwards the authentication to the domain controller.

There are three software components used in this system. Two make up the passthrough system and one resides on the domain controller to generate challenges and accept forwarded responses. The following figure shows the flow of information between the various processes. The client sends the request to the RPC runtime on the server that uses the NTLM Security Support Provider (libsecurity.so) to authenticate the client. The SSP contains the logic to negotiate the authentication level and calls the Local Security Authority (LSA) to authenticate the client. Since the Windows NT LSA is not present on other platforms, the LSA calls are forwarded to the Private Authentication Layer Daemon (PAULAD). The PAULAD service forwards the request to the domain controller by calling the Private Authentication Layer Service (PAULAS) which is running on the domain controller. The PAULAS service has the ability to process login requests or challenge/response requests.

Figure 11-2: PAULAS and PAULAD



In order to secure the communications between the passthrough system and the domain controller, both systems must share a password used to encrypt the communications. This password is stored in a text file on the server and domain controller. After the domain controller performs the authentication, the result is sent back to the PAULAD process. It then passes the response back to the NTLM SSP, which acts on it accordingly.

The next section specifies the NTLM protocol used to authenticate the client. Following sections will detail the communications of the passthrough system and how the two protocols interact.

11.2 NTLM Security

This section defines the NTLM security protocol that is used in this

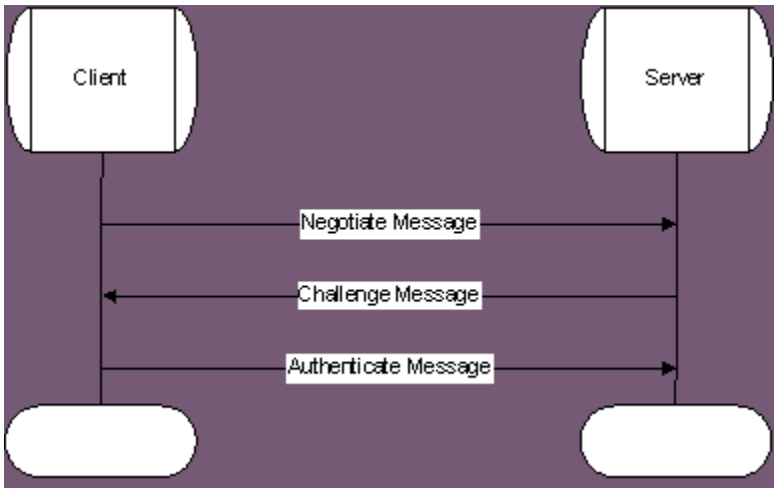
system. Since the passthrough system is opaque to the client, this section will describe the protocol between a client and server.

The protocol will be described first and then the details of the protocol messages will be specified.

11.2.1 Protocol

This section will describe the NTLM protocol. The following figure shows the protocol sequence between the client and server.

Figure 11-3: Protocol Sequence between Client and Server



The protocol is initiated by the client by creating a negotiate message. The Negotiate message contains the SSP signature, the message type, and the negotiate flags. The server responds to the Negotiate message by creating a challenge message. The challenge message contains the SSP signature, the message type, the Negotiate flags, and a challenge.

The client takes the challenge and produces a response. It then creates an authenticate message that contains the SSP signature, the message type, the challenge response, the domain name, the user name, and the optional workstation name.

The server looks at the challenge response and if it is the right response, the authentication is successful.

The following sections describe the messages in detail and the algorithms used to compute their contents.

11.2.2 The Negotiate Message

The client initializes the authentication protocol by generating a Negotiate message. The structure of the Negotiate message follows.

```
typedef struct _NEGOTIATE_MESSAGE {
    UCHAR           Signature[sizeof(NTLMSSP_SIGNATURE)];
    NTLM_MESSAGE_TYPE MessageType;
    ULONG           NegotiateFlags;
} NEGOTIATE_MESSAGE, *PNEGOTIATE_MESSAGE;
```

The Signature is a string that specifies the type of Security Support Provider. This string is always ASCII regardless of the string types used in the rest of the protocol. For NTLM the value follows.

```
#define NTLMSSP_SIGNATURE "NTLMSSP"
```

The message type specifies what type of message is being sent. Each of the three protocol messages has a type defined for it.

```
#define NtLmNegotiate     1
#define NtLmChallenge     2
#define NtLmAuthenticate  3
#define UnknownMessage    8
```

The Negotiate message uses the Message Type `NtLmNegotiate`. The `NegotiateFlags` field tells the server which options the client requires. The Negotiate flags are as follows.

```
   #define NTLMSSP_NEGOTIATE_UNICODE       0x01  // Text
strings are in unicode
   #define NTLMSSP_NEGOTIATE_OEM           0x02  // Text
strings are in OEM
   #define NTLMSSP_REQUEST_TARGET          0x04  // Server
return its auth realm
   #define NTLMSSP_NEGOTIATE_SIGN        0x0010  // Request
signature capability
   #define NTLMSSP_NEGOTIATE_SEAL        0x0020  // Request
confidentiality

   #define NTLMSSP_NEGOTIATE_LM_KEY      0x0080  // Use
LM session key for sign/seal
   #define NTLMSSP_NEGOTIATE_NTLM        0x0200  // NTLM
authentication
   #define NTLMSSP_NEGOTIATE_LOCAL_CALL  0x4000  //
client/server on same machine
   #define NTLMSSP_NEGOTIATE_ALWAYS_SIGN 0x8000  //
Sign for all security levels
```

11.2.2.1 Flag Details

`NTLMSSP_NEGOTIATE_UNICODE` and `NTLMSSP_NEGOTIATE_OEM`

These flags affect the character set of text strings within the protocol. Neither of these flags affects the Signature field, it is always ASCII. The `NT/UNIX` implementations set both of these flags to indicate it can send text in either format. The DOS/WIN16/MAC implementations will only set the `NTLMSSP_NEGOTIATE_OEM` flag.

`NTLMSSP_REQUEST_TARGET`

This flag is set by a client to request the authentication realm of the server. The server's authentication realm specifies where the authentication is occurring. The authentication can occur at the server or at its domain. None of the NT/UNIX/DOS/WIN16/MAC clients set this flag.

`NTLMSSP_NEGOTIATE_SIGN` and `NTLMSSP_NEGOTIATE_SEAL`

The `NTLMSSP_NEGOTIATE_SIGN` flag specifies the messages generated should all contain a digital signature. The `NTLMSSP_NEGOTIATE_SEAL` flag specifies the messages generated should be encrypted prior to being sent.

`NTLMSSP_NEGOTIATE_LM_KEY`

This flag is set if either side wants message sequence or replay detection. Message sequence and replay detection protects the server from a user recording network packets and replaying them.

`NTLMSSP_NEGOTIATE_NTLM`

This flag specifies the authentication protocol to use. This flag is always set.

`NTLMSSP_NEGOTIATE_LOCAL_CALL`

The server may specify this flag if the client and server are running on the same machine. If the client sets this flag, it is ignored.

`NTLMSSP_NEGOTIATE_ALWAYS_SIGN`

This flag indicates the client would like to sign each message with a dummy signature. This flag is ignored if the `NTLMSSP_NEGOTIATE_SIGN` and `NTLMSSP_NEGOTIATE_SEAL` flags are set.

11.2.3 The Challenge Message

In response to a negotiate message, the server will generate a challenge message. The challenge message contains a signature, a message type, the target name, a negotiated capability flags word, and a challenge as follows.

```
typedef _CHALLENGE_MESSAGE {
    UCHAR           Signature[sizeof(NTLM_SIGNATURE)];
    NTLM_MESSAGE_TYPE MessageType;
    STRING          TargetName;
    ULONG           NegotiateFlags;
    UCHAR
Challenge[MSV1_0_CHALLENGE_LENGTH];
} CHALLENGE_MESSAGE, *PCHALLENGE_MESSAGE;
```

The Signature is a string that specifies the type of Security Support Provider. This string is always ASCII regardless of the string types used in the rest of the protocol. For NTLM the value follows.

```
#define NTLMSSP_SIGNATURE "NTLMSSP"
```

The Challenge message uses the Message Type `NtLmChallenge`. The `TargetName` field contains the authentication realm of the server if the `NTLMSSP_REQUEST_TARGET` flag is set in the negotiate message. The `NegotiateFlags` field tells the client which options the server requires. The Negotiate flags for the server are the same as for the client with the addition of the following.

```
#define NTLMSSP_TARGET_TYPE_DOMAIN  0x10000
#define NTLMSSP_TARGET_TYPE_SERVER  0x20000
```

11.2.3.1 Flag Details

`NTLMSSP_NEGOTIATE_UNICODE`
: If the client sets this flag, then it will be set in the challenge message by the NT implementation otherwise `NTLMSSP_NEGOTIATE_OEM` will be set.

`NTLMSSP_REQUEST_TARGET`
: If the client sets this flag, then the `TargetName` field will be filled in. Otherwise, the `TargetName` field is an empty string. The rules for determining the target name follow.

- The `TargetName` value for an NT Server will be the domain name of the system.
- The `TargetName` for an NT Workstation will be the domain name in which the workstation is a member.
- The `TargetName` for an NT Workstation that is not in a domain will be the workstation name without the leading ``\\''. If this flag is set in the negotiate message it must be set in the challenge message.

`NTLMSSP_TARGET_TYPE_DOMAIN`
: This flag is set if the target type is a domain controller.

`NTLMSSP_TARGET_TYPE_SERVER`
: This flag is set if the target type is an NT server.

See Section Section 11.2.3 for details on how to calculate the challenge field.

11.2.4 The Authenticate Message

After initializing the authentication protocol, the server challenges the client. The client in response generates an authenticate message. This message contains a signature, a message type,

the optional LM challenge response, and the optional NT challenge response, the optional domain name, the user name, and the optional workstation name as follows.

```
typedef _AUTHENTICATE_MESSAGE {
    UCHAR           Signature[sizeof(NTLM_SIGNATURE)];
    NTLM_MESSAGE_TYPE MessageType;
    SECURITY_BUFFER   LmChallengeResponse;
    SECURITY_BUFFER   NtChallengeResponse;
    SECURITY_BUFFER   DomainName;
    SECURITY_BUFFER   UserName;
    SECURITY_BUFFER   Workstation;
    SECURITY_BUFFER   SessionKey;
    ULONG           NegotiateFlags;
} AUTHENTICATE_MESSAGE,
*PAUTHENTICATE_MESSAGE;
```

The Authenticate message contains a new type `SECURITY_BUFFER`. This type is defined as follows.

```
typedef struct _SECURITY_BUFFER {
    USHORT Length;
    USHORT MaximumLength;
    ULONG  Buffer;
} SECURITY_BUFFER;
```

The length field specifies the length of the data in the Buffer. The `MaximumLength` field specifies the total allocated length of the Buffer. The Buffer field contains a byte offset of the data from the beginning of the message. If the Buffer contains a `NULL` string, then the Length, `MaximumLength` and Buffer fields will all be set to zero. If the string is a `UNICODE` value then the Length and Buffer

fields must be a multiple of 2.

The Signature is a string that specifies the type of Security Support Provider. This string is always ASCII regardless of the string types used in the rest of the protocol. For NTLM the value follows.

```
#define NTLMSSP_SIGNATURE "NTLMSSP"
```

The Authenticate message uses the Message Type `NtLmAuthenticate`. The `LmChallengeResponse` contains the response to the Challenge field in the Challenge message. See Section Section 11.2.7 for details of how to calculate this value.

The `NtChallengeResponse` is not used and can be set to `NULL`. If the `NTLMSSP_NEGOTIATE_UNICODE` was set then the `DomainName`, `UserName`, and Workstation fields should be `UNICODE` text strings.

The `DomainName` field is the name of the domain containing the users account. If the users account is on a workstation, then the `DomainName` should be the name of the workstation the user originally logged on to. If the `DomainName` is `NULL`, there will be a performance penalty because NT will find users proper domain.

The `UserName` field contains the name of the user to be authenticated. The Workstation field is the name of the computer the caller originally logged into. This field can be `NULL`.

The `SessionKey` and `NegotiateFlags` are used only during datagram authentication and will not be specified here. In connection oriented authentication, the client and server calculate the session key independently based on the user password. This session key is never passed across the network.

The maximum size of the Authenticate message can be calculated by the following macro.

```
   #define NTLMSSP_MAX_MESSAGE_SIZE
(sizeof(AUTHENTICATE_MESSAGE) +
                              LM_RESPONSE_LENGTH +
                              NT_RESPONSE_LENGTH +
                              (DNLEN + 1) * sizeof(WCHAR) +
                              (UNLEN + 1) * sizeof(WCHAR) +
                              (CNLEN + 1) * sizeof(WCHAR) +

MSV1_0_USER_SESSION_KEY_LENGTH)
```

11.2.5 Reauthentication

If the transport drops the connection, the client can reauthenticate by creating an Authentication message that has all the fields set to `NULL` except the signature and message type fields.

11.2.6 Challenge Algorithm

This section describes the algorithm for generating a challenge message. The challenge field is an 8-byte value used to test the client's identity. The challenge field requires two characteristics, it must be unpredictable and must never repeat.

The algorithm uses a system time as a seed to a random number generator. To protect against duplicate seeds resulting from back to back calls, there is also a count (`cChallenge`) that is incremented each time the function is called.

The WIN32 API function `NtQuerySystemTime` is called to get the system time. It returns the current time in a 64-bit integer. The second and third bytes of this value are used because they change the most frequently. Therefore the seed is calculated as

follows:

```
BYTE SysTime[8];
NtQuerySystemTime(&SysTime);
SysTime += cChallenge;
Seed =((SysTime[1] + 1) <<  0) |
 ((SysTime[2] + 0) <<  8) |
 ((SysTime[3] - 1) << 16) |
 ((SysTime[4] + 0) << 24)
```

We then increment the counter:

```
cChallenge += 0x100;
```

The counter must be incremented by 0x100 because the low byte is ignored in generating the seed.

Next three numbers are randomly generated using `RtlUniform`. Three numbers are needed because `RtlUniform` does not generate negative numbers. The third random number is used to negate the two challenge values.

```
ULONG ulChallenge[2];
ulChallenge[0] = RtlUniform(Seed);
ulChallenge[1] = RtlUniform(Seed);
ulNegate = RtlUniform(Seed);
if (ulNegate & 0x1) ulChallenge[0] |= 0x80000000;
if (ulNegate & 0x2) ulChallenge[1] |= 0x80000000;
```

`ulChallenge` now contains the challenge.

11.2.7 Response Message Algorithm

This section describes the calculation of the response message from the challenge message. The purpose of the response

message is to prove to the server that the client has access to a shared secret, the user password. To protect that password from eavesdroppers, it is not sent across the wire. The client must take the challenge and encrypt the challenge with the user password. It can then send this signature across the wire without fear of compromising the user password. For NTLM, the Data Encryption Standard is used to create a digital signature of the challenge.

To calculate a digital signature, we need the challenge which is 8 bytes (challenge), the `LanMan` password which is 16 bytes (`LMPW`), and we generate three 8-byte signatures (`LMResp`).

Following is a key to the notation used to describe the algorithm.

Table 11-1: Response Message Notation Key

Notation	Meaning
E(a,b)	Encrypt a block (b) using DES with key (a)
Substr(a,b, e)	Generate a substring of (a) starting with the byte at offset (b) and ending with the byte at offset (e).
Fill(b, e, v)	Fill starting at offset (b) and ending at offset (e) with value (v).
Zeros(a)	Fill a with zero (0) values.

We calculate the first signature by encrypting the challenge using the first 7 characters of the LM password:

```
LMResp[0] = E(substr(LMPW[0], 0, 6), challenge)
```

We then calculate the second signature by encrypting the challenge using the next 7 characters of the LM password:

```
LMResp[1] = E(substr(LMPW[0], 7, 13), challenge)
```

We calculate the third signature by creating a key with the last two bytes of the LM password and the rest of the key zero filled:

```
BYTE[7] K;
K= zeros(7)
K=substr(LMPW[0], 14, 15)
LMResp[2] = E(K, challenge)
```

The response is now contained in `LMResp`.

11.2.8 Session Key Algorithm

This section describes the algorithm used to generate a session key. A session key is used if the `NTLMSSP_NEGOTIATE_USE_LM_KEY` is set. This key is used by NTLM when encrypting information sent across the network. NTLM uses the RC4 algorithm to perform this encryption.

To generate the session key, the algorithm for generating a response message is used. See Section Section 11.2.7 for the details of this algorithm. This algorithm takes three parameters, an 8-byte challenge, a 16-byte key, and it generates three 8-byte signatures. Following is the prototype for the Challenge/Response function:

```
CalculateChallengeResponse([in]BYTE challenge[8],[in] BYTE
key[16],[out]BYTE LMResp[3][8]);
```

Create the key by placing the user password in the first 8-bytes.

Fill in the remaining 8 bytes with 0xbd:

```
LMKey = substr (password, 0, 7)
LMKey = fill (8, 15, 0xbd)
```

We can then call `CalculateChallengeResponse`. We need to use the `LMResponse` the client generated as the challenge:

```
CalculateChallengeResponse(LMResponse, LMKey,
SessionKey)
```

From this calculation we get three 8-byte values. We disregard the third value. We then set the last three values of the first signature to 0xe5 0x38 0xb0:

```
SessionKey[5] = 0xe5;
SessionKey[6] = 0x38;
SessionKey[7] = 0xb0;
```

The first 16 bytes of session key are now valid.

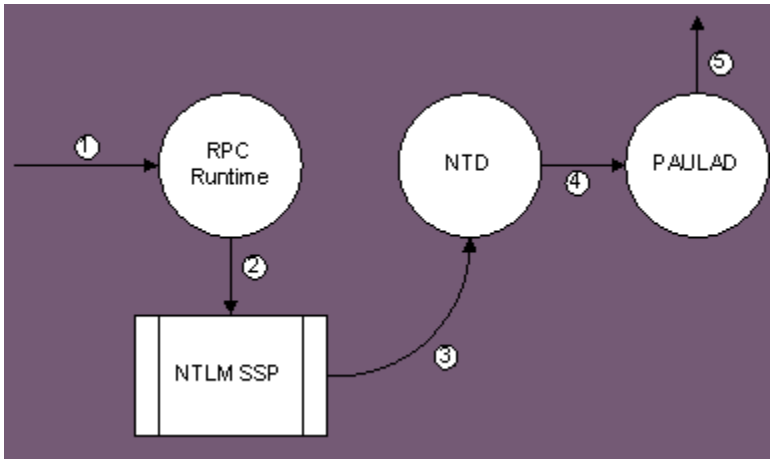
We disregard the third value because RC4 uses a 16-byte key. A 16-byte key has 128 bits. Since the second 8 bytes of the password were 0xbdbdbdbd, the effective key length was 64 bits. Because the three bytes at offsets 5,6,7 were filled in, the effective key length is now 40 bytes. This enables this algorithm to be exported because even though the key is 128 bits, all except 40 bits are well known.

11.3 Client-Passthrough Communications

This section describes the communications between the client through to the PAULAD process. Since the wire protocol is DCOM, it will not be described here. Instead see [DCOM] for information on how to implement the DCOM protocol.

The security calls are passed from the RPC runtime to the NTLM SSP by using the Security Support Provider Interface. See [SSPI] for information on how to implement the SSPI.

Figure 11-4: Client to PAULAD



The call arrives on a RPC port and the runtime pulls the message off the wire. The runtime extracts the security information from the message and passes it to the NTLM SSP. The NTLM SSP in turn calls the Local Security Authority. On Windows NT, the LSA is a component of the operating system. On other platforms, the LSA is implemented in the NT Kernel Daemon (`NTD`) and simply forwards the calls on to the PAULAD process. The method of passing the information is implementation specific and will not be specified here. The LSA calls that must be implemented are as follows.

- `LsaCallAuthenticationPackage`
- `LsaFreeReturnBuffer`
- `LsaLogonUser`
- `LsaLookupAuthenticationPackage`
- `LsaRegisterLogonProcess`
- `LsaDeregisterLogonProcess`

An authentication occurs when the NTLM negotiation is successful.

When the SSP calls `LsaCallAuthenticationPackage`, the call is forwarded to PAULAD to calculate the LM and NT Responses from a challenge and to calculate the user session key and `LanMan` session key.

When the server receives the authenticate message it calls `LsaLogonUser`. This message is sent to PAULAD that then forwards the logon request to the domain controller. The domain controller calls `LsaLogonUser` and returns the result of the operation.

11.4 Passthrough-Domain Communications

This section will describe the communications between the passthrough security system (PAULAD) and the domain controller (PAULAS). Unlike the client-passthrough communications, the passthrough-domain communications uses standard RPC for the communication.

The PAULAS service is a standards RPC server and has an interface with two methods. The `PaulaLMLogonRequest` will perform a `LsaLogonUser` for the passthrough security system. The `PaulaLMChallengeResponse` will generate responses from NTLM challenges.

The IDL declaration for the interface is as follows:

```
[
  uuid (8f529820-168e-11d1-b78e-00a0242a78d8),
  version(1.0),
```

```
  pointer_default(unique)
]
interface Paula /* Private AUthentication LAyer */
{
 ...
}
```

The `PaulaLMLogonRequest` IDL is as follows:

```
void PaulaLMLogonRequest (
    [in] unsigned int CookieIn,
    [in] unsigned char Domain[64],
    [in] unsigned char Workstation[64],
    [in] unsigned char UserName[64],
    [in] unsigned char Challenge[8],
    [in] unsigned char CaseInsensitiveChallengeResponse[24],
    [in] unsigned char CaseSensitiveChallengeResponse[24],
    [out] unsigned int *CookieOut,
    [out] unsigned int *Status,
    [out] unsigned int *SubStatus,
    [out] unsigned int *Result,
    [out] unsigned char UserSessionKey[16],
    [out] unsigned char LanmanSessionKey[8]);
```

The input parameters `CookieIn`, Domain, Workstation, `UserName`, `Challenge`, `CaseInsensitiveChallengeResponse`, and `CaseSensitiveChallengeResponse` are encrypted with RC4 and the password stored in the password file before being sent across the wire. Likewise, the output parameters, `CookieOut`, `Status`, `SubStatus`, `Result`, `UserSessionKey`, and `LanmanSession` key must be decrypted using RC4 and the password stored in the password file before being used.

The `PaulaLMChallengeResponseRequest` IDL is as follows:

```
void PaulaLMChallengeResponseRequest (
    [in] unsigned int CookieIn,
    [in] unsigned int ParameterControl,
    [in] unsigned char Domain[64],
    [in] unsigned char Workstation[64],
    [in] unsigned char UserName[64],
    [in] unsigned int LogonIdLSW,
    [in] unsigned int LogonIdMSW,
    [in] unsigned char LmChallenge[8],
    [in] unsigned char NtChallenge[8],
    [out] unsigned int *CookieOut,
    [out] unsigned int *Status,
    [out] unsigned int *ProtocolStatus,
    [out] unsigned char CaseInsensitiveChallengeResponse[24],
    [out] unsigned char CaseSensitiveChallengeResponse[24],
    [out] unsigned char UserNameRet[64],
    [out] unsigned char LogonDomainNameRet[64],
    [out] unsigned char UserSessionKey[16],
    [out] unsigned char LanmanSessionKey[8]);
```

The input parameters, `CookieIn`, `ParameterControl`, `Domain`, `Workstation`, `UserName`, `LogonIdLSW`, `LogonIdMSW`, `LmChallenge`, and `NtChallenge` are encrypted with RC4 and the password stored in the password file before being sent across the wire. Likewise, the output parameters, `Status`, `ProtocolStatus`, `CaseInsensitiveChallengeResponse`, `CaseSensitiveChallengeResponse`, `UserNameRet`, `LogonDomainNameRet`, `UserSessionKey`, and `LanmanSessionKey` must be decrypted using RC4 and the password stored in the

password file before being used.

11.5 Bibliography

[DCOM] Distributed Component Object Model Protocol--DCOM/1.0. Nat Brown, Charlie Kindel. Microsoft Corporation January 1998. This draft specification is available through the Internet Engineering Task Force site at www.internic.net. It can also be downloaded directly from Microsoft.

[SSPI] Microsoft® Windows NT® Security Support Provider Interface. © 1996 Microsoft Corporation.
