[MS-CSSP]:
Credential Security Support Provider (CredSSP) Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
12/18/2006	0.1		MCPP Milestone 2 Initial Availability
03/02/2007	1.0		MCPP Milestone 2
04/03/2007	1.1		Monthly release
05/11/2007	1.2		Monthly release

Date	Revision History	Revision Class	Comments
06/01/2007	1.2.1	Editorial	Revised and edited the technical content.
07/03/2007	1.2.2	Editorial	Revised and edited the technical content.
07/20/2007	1.2.3	Editorial	Revised and edited the technical content.
08/10/2007	1.2.4	Editorial	Revised and edited the technical content.
09/28/2007	1.2.5	Editorial	Revised and edited the technical content.
10/23/2007	1.3	Minor	Updated the technical content.
11/30/2007	1.3.1	Editorial	Revised and edited the technical content.
01/25/2008	1.3.2	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

The CredSSP Protocol enables an application to securely delegate a user's **credentials** from a client to a target server. This protocol first establishes an encrypted channel between the client and the target server by using **Transport Layer Security (TLS)** (as specified in [RFC2246]). The CredSSP Protocol uses TLS as an encrypted pipe; it does not rely on the client/server authentication services that are available in TLS. The CredSSP Protocol then uses the Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions to negotiate a **Generic Security Services (GSS)** mechanism that performs **mutual authentication** and GSS confidentiality services to securely bind to the TLS channel and encrypt the credentials for the target server. It should be noted that all GSS security tokens are sent over the encrypted TLS channel.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Application Protocol
Certification Authority (CA)
Credential
Domain
Generic Security Services (GSS)
Kerberos
Mutual Authentication
NT LAN Manager Protocol (NTLM)
Public Key Infrastructure (PKI)
Security Protocol
Service Principal Name (SPN)
Transport Layer Security (TLS)
Trust

The following terms are specific to this document:

CredSSP Client: Any application that executes the role of the client as prescribed by the CredSSP Protocol described in this document.

CredSSP Server: Any application that executes the role of the server as prescribed by the CredSSP Protocol described in this document.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-KILE] Microsoft Corporation, "Kerberos Protocol Extensions", January 2007.

[MS-NLMP] Microsoft Corporation, "NT LAN Manager (NTLM) Authentication Protocol Specification", June 2007.

[MS-SPNG] Microsoft Corporation, "Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions", January 2007.

[RFC2078] Linn, J., "Generic Security Service Application Program Interface, Version 2", RFC 2078, January 1997, http://www.ietf.org/rfc/rfc2078.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC2246] Dierks, T. and Allen, C., "The TLS Protocol Version 1.0", RFC 2246, January 1999, http://www.ietf.org/rfc/rfc2246.txt

[RFC3280] Housley, R., Polk, W., Ford, W., and Solo, D., "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002, http://www.ietf.org/rfc/rfc3280.txt

[RFC4120] Neuman, C., Yu, T., Hartman, S., and Raeburn, K., "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005, http://www.ietf.org/rfc/rfc4120.txt

[RFC793] Postel, J., "Transmission Control Protocol: DARPA Internet Program Protocol Specification", RFC 793, September 1981, http://www.ietf.org/rfc/rfc0793.txt

[X680] ITU-T, "Abstract Syntax Notation One (ASN.1): Specification of Basic Notation", Recommendation X.680, July 2002, http://www.itu.int/rec/T-REC-X.680/en

Note There is a charge to download the specification.

[X690] ITU-T, "Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", Recommendation X.690, July 2002, http://www.itu.int/rec/T-REC-X.690/en

Note There is a charge to download the specification.

1.2.2 Informative References

This specification contains no informative references.

1.3 Protocol Overview (Synopsis)

The Credential Security Support Provider (CredSSP) Protocol enables an application to securely delegate a user's credentials from a client to a target server. For example, the Microsoft Terminal Server uses the CredSSP Protocol to securely delegate the user's password or smart card PIN from the client to the server to remotely log on the user and establish a terminal services session.

Policy settings control whether a client delegates the user's credentials in order to assure that the user's credentials are not delegated to an unauthorized server (a computer under the administrative control of an attacker). Although **trust** may exist to facilitate authentication between the client and server, it does not mean that the target server is trusted with the user's credentials.<1> For example, trust may be based on the **Kerberos** Protocol [RFC4120] or **NTLM** [MS-NLMP].

The CredSSP Protocol is a composite protocol that relies on other standards-based **security protocols**. It first uses the Transport Layer Security Protocol (TLS) to establish an encrypted

channel between the **CredSSP client** and the **CredSSP server**. (The client is anonymous at this point; the client and the server may have no common trusted **certification authority** root.)

All subsequent messages are sent over this channel. The CredSSP Protocol then uses the Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) to authenticate the user and server in the encrypted TLS session. (SPNGO is specified in [MS-SPNG].)

SPNEGO provides a framework for two parties that are engaged in authentication to select from a set of possible authentication mechanisms. This framework provides selection in a manner that preserves the opaque nature of the security protocols to the **application protocol** that uses SPNEGO, in this case, CredSSP Protocol.

The CredSSP Protocol uses SPNEGO to mutually authenticate the CredSSP client and CredSSP server. It then uses the encryption key that is established under SPNEGO to securely bind to the TLS session (the process by which the server's public key that is used in the TLS handshake is authenticated). The client encrypts the server's public key by using the encryption key that is established under SPNEGO and sends it to the server. The server verifies that it is the same public key that was used in the TLS handshake and sends an acknowledgment (also encrypted under the SPNEGO encryption key) back to the client. (For more information about this step, see section 3.1.) Lastly, the client sends the user's credentials, which are encrypted under the SPNEGO encryption key, to the server.

All subsequent data that may be sent between the client and server application by using the CredSSP Protocol is encrypted under TLS. The only new on-the-wire formats that are introduced by the CredSSP Protocol are the encapsulation of the SPNEGO tokens sent over the TLS channel, the binding between the TLS and SPNEGO protocols, and the format of the user credentials.

1.4 Relationship to Other Protocols

The CredSSP Protocol uses the TLS protocol, as specified in [RFC2246], to encrypt all traffic between the CredSSP client and CredSSP server. The TLS protocol requires a reliable transport, such as TCP (as specified in [RFC793]), for all messages that are exchanged between the client and the server.

The CredSSP Protocol uses SPNEGO [MS-SPNG] for mutual authentication between the CredSSP client and CredSSP server. SPNEGO requires at least one other Generic Security Services (GSS) [RFC2078] compatible authentication protocol (in addition to SPNEGO itself) to be present for it to work. SPNEGO has no dependence on any specific GSS-compatible protocols; however, the Kerberos Protocol [MS-KILE] is typically used.<2>

The Remote Desktop Protocol (RDP) uses the CredSSP Protocol to delegate credentials from the RDP client to the RDP server and to encrypt all data that follows by using the TLS channel that is established as part of the CredSSP Protocol.

1.5 Prerequisites/Preconditions

The CredSSP Protocol assumes the following:

- The CredSSP client MUST have access to the user's credentials (the CredSSP Protocol delegates these credentials to the CredSSP server).<3>
- A source of cryptographically useful random numbers MUST be available on the client and server for generating a nonce that is used by the TLS Protocol.
- The CredSSP server MUST have an X509 certificate (as specified in [RFC3280]), for use in TLS. The certificate may be self-signed or issued by a third-party certification authority. The CredSSP

Protocol does not assume a common certification authority root between the client and the server.

- The CredSSP Protocol uses the SPNEGO protocol for mutual client/server authentication; at least one other GSS-compatible authentication protocol, in addition to the CredSSP Protocol, MUST be present for it to work.<4>

1.6 Applicability Statement

CredSSP delegates the user's credentials from a client to a server over a mutually authenticated encrypted channel. To avoid revealing the user credentials to unauthorized hosts, the CredSSP client should delegate only to trusted servers, as expressed through the security policy that governs the client's computer. It should be noted that CredSSP was designed to enable the server to impersonate the client across a number of different applications that require the user's long-lived credentials (password).

1.7 Versioning and Capability Negotiation

Versioning and capability negotiation are supported in the CredSSP Protocol as follows:

- Protocol Versions: The CredSSP Protocol supports versioning (the version field of the TSRequest structure, section 2.2.1); however, version 2.0 is currently the only available version.
- Security and Authentication Methods: The CredSSP Protocol uses the SPNEGO protocol to negotiate the underlying authentication mechanism. Similarly, the CredSSP Protocol relies on the TLS protocol to negotiate the cryptographic algorithms that are used for channel confidentiality and integrity.
- Localization: The CredSSP Protocol is not localization-dependent.

1.8 Vendor-Extensible Fields

The CredSSP Protocol does not have any vendor-extensible fields.

1.9 Standards Assignments

The CredSSP Protocol does not have any standards assignments. Standards assignments for the SPNEGO and TLS Protocols are specified in [MS-SPNG] section 1.9 and [RFC2246] section G, respectively.

2 Messages

2.1 Transport

Because the CredSSP Protocol uses TLS, it requires that all messages that are exchanged between the client and server are transmitted by using a reliable transport protocol, such as TCP (as specified in [RFC793]).<5>

2.2 Message Syntax

The CredSSP Protocol introduces the TSRequest message. The client and server use this message to encapsulate the SPNEGO tokens and TSCredentials message that the client uses to delegate the user's credentials to the CredSSP server over a TLS connection. These messages are encoded by using ASN.1 (as specified in [X690]) and Distinguished Encoding Rules (DER).

2.2.1 TSRequest

The **TSRequest** structure is the top-most structure that is used by the CredSSP client and CredSSP server. It contains the SPNEGO messages between the client and server, and either the public key authentication messages that are used to bind to the TLS session or the client credentials that are delegated to the server. The **TSRequest** message is always sent over the TLS encrypted channel between the client and server in a CredSSP Protocol exchange (see step 1 in section 3.5).

```
TSRequest ::= SEQUENCE {
        version    [0] INTEGER,
        negoTokens [1] NegoData  OPTIONAL,
        authInfo   [2] OCTET STRING OPTIONAL,
        pubKeyAuth [3] OCTET STRING OPTIONAL
}
```

version: This field specifies the supported version of the CredSSP Protocol. This field MUST be 2. If the version is greater than 2, a version 2 client or server treats its peer as one that is compatible with version 2 of the CredSSP Protocol.

negoTokens: A NegoData structure, as defined in section 2.2.1.1, that contains the SPNEGO messages that are passed between the client and server.

authInfo: A TSCredentials structure, as defined in section 2.2.1.2, that contains the user's credentials that are delegated to the server. The **authinfo** field MUST be encrypted under the encryption key that is negotiated under the SPNEGO package.

pubKeyAuth: This field is used to assure that the public key that is used by the server during the TLS handshake belongs to the target server and not to a "man in the middle." This TLS session-binding is described in section 3.5. After the client completes the SPNEGO phase of the CredSSP Protocol, it uses GSS confidentiality services and encrypts the server's public key by using the encryption key that is negotiated under SPNEGO (K_{spnego}). The **pubKeyAuth** field carries the encrypted public key to the server. In response, the server uses the **pubKeyAuth** field to transmit to the client a modified version of the public key (as described in section 3.5) that is encrypted under the encryption key that is negotiated under SPNEGO.

2.2.1.1 NegoData

The **NegoData** structure contains the SPNEGO messages, as specified in [MS-SPNG] section 2.

```
NegoData ::= SEQUENCE OF SEQUENCE {
        negoToken [0] OCTET STRING
}
```

NegoToken: One or more SPNEGO tokens, as specified in [MS-SPNG].<6>

2.2.1.2 TSCredentials

The TSCredentials structure contains both the user's credentials that are delegated to the server and their type.

```
TSCredentials ::= SEQUENCE {
        credType    [0] INTEGER,
        credentials [1] OCTET STRING
}
```

credType: Defines the type of credentials that are carried in the **credentials** field. credType MUST be one of the following values:

Value	Meaning
1	**credentials** contains a TSPasswordCreds structure that defines the user's password credentials.
2	**credentials** contains a TSSmartCardCreds structure that defines the user's smart card credentials.

credentials: Contains either the user's password or smart card credentials in either a TSPasswordCreds structure or TSSmartCardCreds structure.

2.2.1.2.1 TSPasswordCreds

The **TSPasswordCreds** structure contains the user's password credentials that are delegated to the server.

```
TSPasswordCreds ::= SEQUENCE {
        domainName  [0] OCTET STRING,
        userName    [1] OCTET STRING,
        password    [2] OCTET STRING
}
```

domainName: Contains the name of the user's account **domain**, as specified in [MS-GLOS].

userName: Contains the user's account name.

Password: Contains the user's account password.

2.2.1.2.2 TSSmartCardCreds

The TSSmartCardCreds structure contains the user's smart card credentials that are delegated to the server.

```
TSSmartCardCreds ::= SEQUENCE {
        pin         [0] OCTET STRING,
        cspData     [1] TSCspDataDetail,
        userHint    [2] OCTET STRING OPTIONAL,
        domainHint  [3] OCTET STRING OPTIONAL
}
```

pin: Contains the user's smart card PIN.

cspData: A TSCspDataDetail structure that contains information about the cryptographic service provider (CSP).

userHint: Contains the user's account hint.

domainHint: Contains the user's domain name to which the user's account belongs. This name could be entered by the user when the user is first prompted for the PIN.

2.2.1.2.2.1 TSCspDataDetail

The **TSCspDataDetail** structure contains cryptographic service provider (CSP) information that is used during smart card logon.

```
TSCspDataDetail ::= SEQUENCE {
        keySpec       [0] INTEGER,
        cardName      [1] OCTET STRING OPTIONAL,
        readerName    [2] OCTET STRING OPTIONAL,
        containerName [3] OCTET STRING OPTIONAL,
        cspName       [4] OCTET STRING OPTIONAL
}
```

keySpec: Defines the specification of the user's smart card.

cardName: Specifies the name of the smart card.

readerName: Specifies the name of the smart card reader.

containerName: Specifies the name of the certificate container.

cspName: Specifies the name of the cryptographic service provider.

3 Protocol Details

3.1 Abstract Data Model

The CredSSP Protocol requires the client to perform a policy check to verify that the target server is trusted to receive the user's credentials.

3.2 Timers

There are no timers in the CredSSP Protocol.

3.3 Initialization

There are no changes to the initialization of TLS and SPNEGO, as specified in [RFC2246] and [MS-SPNG], respectively.

3.4 Higher-Layer Triggered Events

The CredSSP Protocol is triggered by a higher-layer application protocol, such as Remote Desktop Protocol (RDP), for delegating the user's credentials to the target server.

3.5 Message Processing Events and Sequencing Rules

The CredSSP Protocol is carried out in the following sequence and is subject to the protocol rules that are described in the following steps.

1. The CredSSP client and CredSSP server first complete the TLS handshake, as specified in [RFC2246]. After the handshake is complete, all subsequent CredSSP Protocol messages are encrypted by the TLS channel. The CredSSP Protocol does not extend the TLS wire protocol. As part of the TLS handshake, the CredSSP server does not request the client's X.509 certificate (thus far, the client is anonymous). Also, the CredSSP Protocol does not require the client to have a commonly trusted certification authority root with the CredSSP server. Thus, the CredSSP server MAY use, for example, a self-signed X.509 certificate.<7>

2. Over the encrypted TLS channel, the SPNEGO handshake between the client and server completes mutual authentication and establishes an encryption key that is used by the SPNEGO confidentiality services, as specified in [RFC4178]. All SPNEGO tokens as well as the underlying encryption algorithms are opaque to the calling application (the CredSSP client and CredSSP server). The wire protocol for SPNEGO is specified in [MS-SPNG].

   The SPNEGO tokens exchanged between the client and the server are encapsulated in the **negoTokens** field of the TSRequest structure. Both the client and the server use this structure as many times as necessary to complete the SPNEGO exchange.<8>

   Note During this phase of the protocol, the OPTIONAL **authInfo** field is omitted from the TSRequest structure by the client and server; and the OPTIONAL **pubKeyAuth** field is omitted by the client unless the client is sending the last SPNEGO token. If the client is sending the last SPNEGO token, the TSRequest structure MUST have both the **negoToken** and the **pubKeyAuth** fields filled in.

3. The client encrypts the public key it received from the server (contained in the X.509 certificate) in the TLS handshake from step 1, by using the confidentiality support of SPNEGO. The public key that is encrypted is the ASN.1 encoded SubjectPublicKeyInfo field from the X.509 certificate, as specified in [RFC3280] section 4.1. The encrypted key is encapsulated in the **pubKeyAuth** field of the TSRequest structure and is sent over the TLS channel to the server.

Note During this phase of the protocol, the OPTIONAL **authInfo** field is omitted from the TSRequest structure; the client MUST send its last SPNEGO token to the server in the **negoTokens** field (see step 2 above) along with the encrypted public key in the **pubKeyAuth** field.

4. After the server receives the public key in step 3, it first verifies that it has the same public key that it used as part of the TLS handshake in step 1. The server then adds 1 to the most significant byte of the public key (the ASN.1 representation of SubjectPublicKeyInfo field, as described in step 3) and encrypts the result by using the SPNEGO encryption services. The encrypted result is encapsulated in the **pubKeyAuth** field of the TSRequest structure, and is sent over the encrypted TLS channel to the client. The addition of 1 to the most significant byte of the public key is performed so that the client-generated **pubKeyAuth** message cannot be replayed back to the client by an attacker.

   Note During this phase of the protocol, the OPTIONAL **authInfo** and **negoTokens** fields are omitted from the TSRequest structure.

5. After the client decrypts the result from 4 and verifies server authenticity by comparing the decrypted key to the public key from the server's X.509 certificate (as specified in [RFC3280], section 4.1), it encrypts the user's credentials (either password or smart card PIN) by using the SPNEGO encryption services. The result is encapsulated in the **authInfo** field of the TSRequest structure and sent over the encrypted TLS channel to the server.

   The TSCredentials structure within the **authInfo** field of the TSRequest structure MAY contain either a TSPasswordCreds or TSSmartCardCreds structure, but MUST NOT contain both.

   Note During this phase of the protocol, the OPTIONAL **pubKeyAuth** and **negoTokens** fields are omitted from the TSRequest structure.

3.6 Timer Events

There are no timer events for the CredSSP Protocol.

3.7 Other Local Events

There are no other local events that impact the operation of this protocol.

4 Protocol Examples

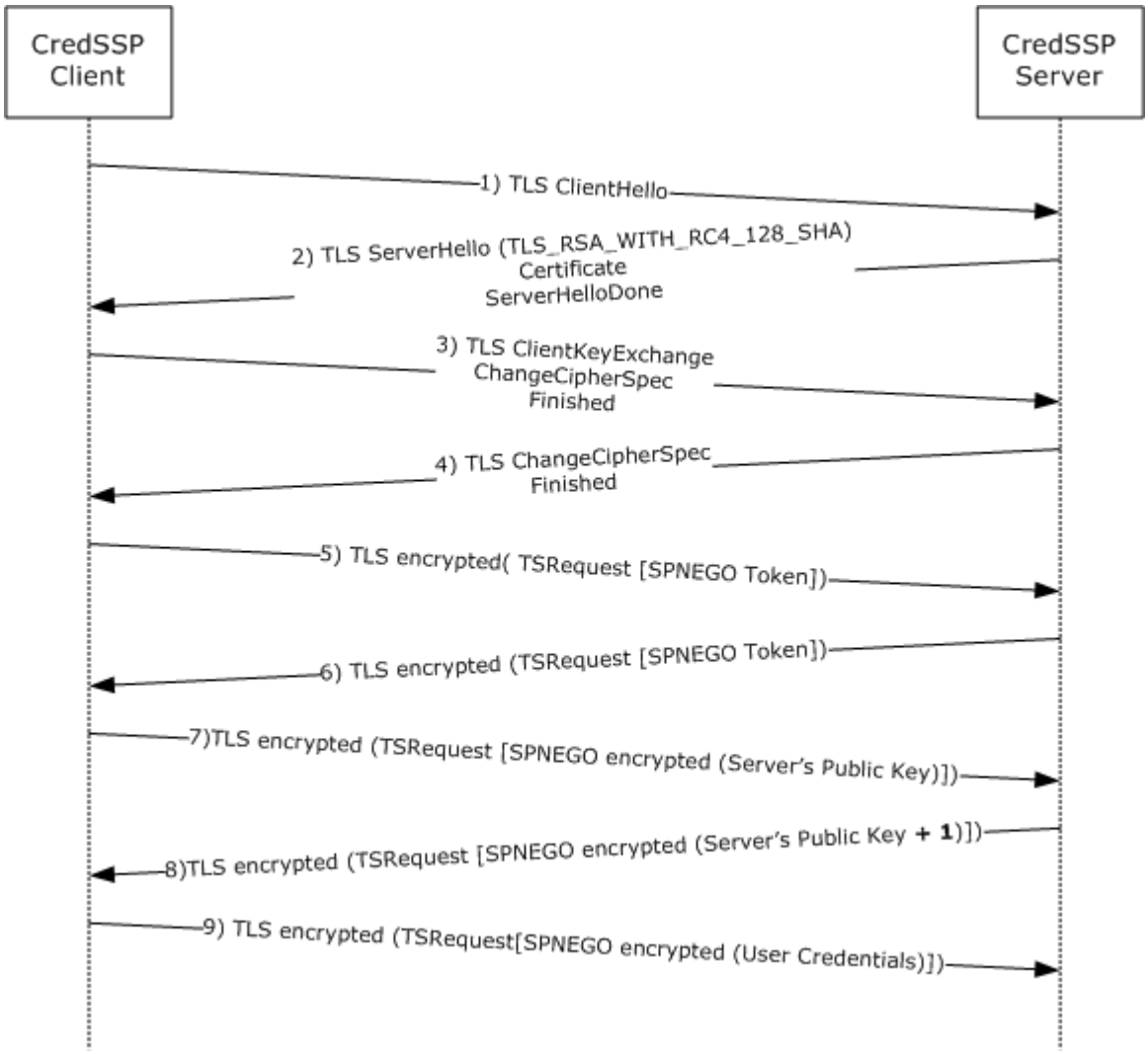

Figure 1: CredSSP negotiation sequence

Steps 1 through 4: The CredSSP client and CredSSP server complete the TLS handshake. When the handshake is complete, all subsequent CredSSP Protocol messages are encrypted by the TLS channel, as specified in [RFC2246]. As part of the TLS handshake, the CredSSP server does not request the client's X.509 certificate (thus far, the client is anonymous). Furthermore, the CredSSP Protocol does not require the client to have a commonly trusted certification authority root with the CredSSP server.

Steps 5 and 6: Over the encrypted TLS channel, the SPNEGO handshake between the client and server completes mutual authentication and establishes an encryption key.

Steps 7 and 8: The public key from the server's X.509 certificate in the TLS handshake is verified that it belongs to the server (and not to a "man in the middle" attacker).

Step 9: The client sends its credentials to the target server that is protected under SPNEGO and TLS encryption.

5 Security

5.1 Security Considerations for Implementers

Be certain that the user credentials that are delegated by the CredSSP Protocol do not fall under an attacker's control. The purpose of the CredSSP Protocol policy settings is to ensure that the user's credentials are not delegated to an unauthorized server (a computer under the administrative control of an attacker). Trust might exist based on **PKI**, the Kerberos Protocol, or NTLM. Although trust may exist to facilitate authentication between the client and server, this does not mean that the target server is trusted with the user's credentials. The policy settings define which servers are trusted with the user's credentials (each setting is a list of **SPNs**; wildcards are allowed). The CredSSP Protocol (client side) only delegates the user's credentials if the server is authenticated to the client and the server's SPN passes a policy check.

Consider the following factors when constructing a security policy for delegating credentials: the security strength of the authentication mechanisms negotiated under SPNEGO (for example, the Kerberos protocol versus NTLM) and the method by which the CredSSP client obtained the user's credentials (Fresh, Saved or Default):

- Fresh Credentials: A calling application that prompts the user and passes in the user's credentials to the CredSSP client.
- Saved Credentials: User credentials that are stored on the local computer for use only with the target server.
- Default Credentials: Credentials that are entered by the user when the user first logs on to the operating system.

The model for delegating the user's credentials can be constructed in terms of the following parameters; each parameter is a list of zero or more service principal names that represent the target server:

- AllowDefCredentials: The password may be passed to the listed targets when authenticating by using the default credentials.
- AllowSavedCredentials: The password may be passed to the listed targets when authenticating by using the saved credentials.
- AllowFreshCredentials: The password may be passed to the target when authenticating by using the fresh credentials.
- DenySavedCredentials: The password may not be passed to the listed targets when authenticating by using the saved credentials.
- DenyFreshCredentials: The password may not be passed to the listed targets when authenticating by using the fresh credentials.
- DenyDefaultCredentials: The password may not be passed to the listed targets when authenticating by using the fresh credentials.
- AllowDefCredentialsWhenNTLMOnly: If the only authentication was NTLM and the user authenticated by using the default credentials, allow the password to be passed to the listed targets.
- AllowSavedCredentialsWhenNTLMOnly: If the only authentication was NTLM and the user authenticated by using saved credentials, allow the password to be passed to the listed targets.

- AllowFreshCredentialsWhenNTLMOnly: If the only authentication was NTLM and the user authenticated by using fresh credentials, allow the password to be passed to the listed targets.

5.2 Index of Security Parameters

There are no security parameters in the CredSSP Protocol.

6 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Vista

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1.3: In Windows Vista, the policy settings for the CredSSP client are expressed in terms of service principal names (SPNs), which define the servers that the client is allowed to send the user's credentials to.

<2> Section 1.4: By default, SPNEGO has the Kerberos protocol and NTLM available, as specified in [MS-NLMP]. The interface for authentication protocols on Windows Vista is open and extensible. Other protocols may be installed on a specific system by third parties, and other protocols may be added as defaults in future versions of Windows.

<3> Section 1.5: In Windows Vista, the CredSSP client first checks if the user's credentials were passed in by the calling application. If so, these credentials are used by the client. If no credentials were passed in by the calling application, the CredSSP Protocol uses credentials that are stored locally in the credentials manager that is associated with the target server. If no credentials are available for the target server, the CredSSP client uses the user's default credentials, which are entered when the user first logs on to the operating system.

<4> Section 1.5: In Windows Vista, the SPNEGO client negotiates Kerberos or NTLM. The Kerberos Protocol is always preferred over NTLM. NTLM is only negotiated if one or both parties do not support the Kerberos Protocol, as specified in [MS-NLMP] section 1.5 and in [MS-KILE].

<5> Section 2.1: The Windows Vista component that implements the CredSSP Protocol is transport-independent—it simply returns opaque CredSSP data back to the calling application. It is up to the calling application to send this CredSSP Protocol data over a reliable transport to its CredSSP Protocol peer.

<6> Section 2.2.1.1: This contains all Kerberos- or NTLM-specific messages as negotiated by SPNEGO.

<7> Section 3.5: In Windows Vista, the CredSSP server can be configured by using any X.509 certificate that is trusted by the client based on a commonly trusted CA root or by using a self-signed certificate.

<8> Section 3.5: The Kerberos or NTLM authentication package is negotiated by SPNEGO. Therefore, the encryption key that is established under SPNEGO is either a Kerberos sub-session key or an NTLM session key that is shared by both sides upon completion of the SPNEGO exchange.

7 Index