[MS-SPNG]:
Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions

---

This protocol documentation is intended for use in conjunction with publicly available standard specifications, network programming art, and Microsoft Windows distributed systems concepts, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

A protocol specification does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them.

Revision Summary

Date	Revision History	Revision Class	Comments
10/22/2006	0.01		MCPP Milestone 1 Initial Availability
01/19/2007	1.0		MCPP Milestone 1
03/02/2007	1.1		Monthly release

Date	Revision History	Revision Class	Comments
04/03/2007	1.2		Monthly release
05/11/2007	1.3		Monthly release
06/01/2007	1.3.1	Editorial	Revised and edited the technical content.
07/03/2007	1.3.2	Editorial	Revised and edited the technical content.
07/20/2007	1.3.3	Editorial	Revised and edited the technical content.
08/10/2007	2.0	Major	Updated and revised the technical content.
09/28/2007	3.0	Major	Updated and revised the technical content.
10/23/2007	4.0	Major	Added technical clarifications.
11/30/2007	5.0	Major	Updated and revised the technical content.
01/25/2008	5.0.1	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

The Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions is an extension to [RFC4178] that specifies a negotiation mechanism for the **Generic Security Service** Application Program Interface (GSS-API), as specified in [RFC2743]. Microsoft Windows 2000 uses the Simple and Protected Generic Security Service Application Programming Interface (GSS-API) Negotiation Mechanism (SPNEGO), as specified in [RFC4178], to select among possible authentication protocols. SPNEGO provides a framework for two parties that are engaged in authentication to select from a set of possible authentication mechanisms, in a manner that preserves the opaque nature of the **security protocols** to the **application protocol** that uses SPNEGO. SPNEGO was first defined in [RFC2478], which has been superseded by [RFC4178].

Several of the extensions or behavior variants that Microsoft made when first implementing SPNEGO against [RFC2478] have been incorporated into [RFC4178]. This specification further explains and documents those variations in the context of the current **remote procedure call (RPC)**, as specified in [RFC4178].

1.1 Glossary

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

Application Protocol
Generic Security Services (GSS)
Object Identifier (OID)
Original Equipment Manufacturer (OEM) Code Page
Remote Procedure Call (RPC)
Security Protocol
Security Token

The following terms are specific to 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-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

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

[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

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000, http://www.ietf.org/rfc/rfc2743.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

[RFC4121] Zhu, L., Jaganathan, K., and Hartman, S., "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2", RFC 4121, July 2005, http://www.ietf.org/rfc/rfc4121.txt

[RFC4178] Zhu, L., Leach, P., Jaganathan, K., and Ingersoll, W., "The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism", RFC 4178, October 2005, http://www.ietf.org/rfc/rfc4178.txt

[UUKA-GSSAPI] Swift, M., Brezak, J., and Moore, P., "User to User Kerberos Authentication using GSS-API", October 2001, http://www.watersprings.org/pub/id/draft-swift-win2k-krb-user2user-03.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

[HTTPAUTH] Jaganathan, K., Zhu, L., and Brezak, J., "Kerberos based HTTP Authentication in Windows", July 2005, http://tools.ietf.org/html/draft-jaganathan-kerberos-http-01

[KAUFMAN] Kaufman, C., Perlman, R., and M. Speciner, "Network Security: Private Communication in a Public World, Second Edition", Prentice Hall, 2002, ISBN: 0130460192.

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions", January 2007.

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol Specification", July 2007.

[MSDN-CP] Microsoft Corporation, "Code Page Identifiers", http://msdn2.microsoft.com/en-us/library/ms776446.aspx

If you have any trouble finding [MSDN-CP], please check here.

[RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996, http://www.ietf.org/rfc/rfc1964.txt

[RFC2251] Wahl, M., Howes, T., and Kille, S., "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997, http://www.ietf.org/rfc/rfc2251.txt

[RFC2478] Baize, E. and Pinkas, D., "The Simple and Protected GSS-API Negotiation Mechanism", RFC 2478, December 1998, http://www.ietf.org/rfc/rfc2478.txt

1.3 Protocol Overview (Synopsis)

1.3.1 Security Background

SPNEGO is a security protocol. As such, the normative references and this specification use common security-related terms. Every effort has been made to use these terms, such as principal, key, and service, in accordance with their use in [RFC4178].

Anyone who wants to understand the variations between SPNEGO Protocol Extensions and [RFC4178] should have a working knowledge of the Generic Security Service API. Several of the informative references, specifically [KAUFMAN], provide excellent top-level information about Generic Security Services (GSS) and the message flow. [KAUFMAN] also provides an excellent survey of other security protocols and concepts, and it helps to explain the terms of art that this specification uses. For more information, see [KAUFMAN].

Historically, the first GSS security mechanism defined was the Kerberos protocol (for more information, see [RFC1964]). The Kerberos protocol has influenced many other mechanisms; in some cases, it may prove helpful to have an example protocol to compare against. Finally, there are details that are not immediately apparent, as specified in [RFC4178] and [RFC2743].

1.3.2 SPNEGO Synopsis

SPNEGO is a security protocol that uses a GSS-API authentication mechanism. GSS-API is a literal set of functions that include both an API and a methodology for approaching authentication. As specified in [RFC2743], GSS-API and the individual security protocols that correspond to the GSS-API (also shortened to GSS) were developed because of the need to insulate application protocols from the specifics of security protocols as much as possible.

This approach led to a simplified form of interaction between an application protocol and an authentication protocol. In this model, an application protocol is responsible for ferrying discrete, opaque packets that the authentication protocol produces. These packets, which are referred to as **security tokens** by the GSS specifications, implement the authentication process. The application protocol has no visibility into the contents of the security tokens; its responsibility is merely to carry them.

The application protocol in this model first invokes the authentication protocol on the client. The client portion of the authentication protocol creates a security token and returns it to the calling application. The application protocol then transmits that security token to the server side of its connection, embedded within the application protocol. On the server side, the server's application protocol extracts the security token and supplies it to the authentication protocol on the server side. The server authentication protocol can process the security token and possibly generate a response; or it can decide that authentication is complete. If another security token is generated, the application protocol MUST carry it back to the client where the process continues.

This exchange of security tokens continues until one side determines that authentication has failed or both sides decide that authentication is complete. If authentication fails, the application protocol drops the connection and indicates the error. If authentication succeeds, the application protocol can be assured of the identity of the participants as far as the supporting authentication protocol can accomplish. The onus of determining success or failure is on the abstracted security protocol, not the application protocol, which greatly simplifies the application protocol author's task.

After the authentication is complete, session-specific security services may be available. The application protocol can then invoke the authentication protocol to sign or encrypt the messages that are sent as part of the application protocol. The session-specific security services operations are done in much the same way, where the application protocol can indicate which portions of the message are to be encrypted, and the application protocol MUST include a per-message security

token. By signing or encrypting the messages, the application can obtain message privacy and integrity, and detect message loss, out of order delivery and duplication.

Because more than one GSS-compatible authentication protocol exists, determining which protocol to use has become more important. The original GSS design had a static, compile-time binding between the application and the GSS implementation. More recent practice is to support more than one authentication mechanism-even for a single application protocol.

SPNEGO fills this need by presenting a GSS-compatible wrapper to other GSS mechanisms. It securely negotiates among several authentication mechanisms, selecting one for use to satisfy the authentication needs of the application protocol.

Microsoft has certain variations to the SPNEGO protocol: certain errors in early implementations and an optimization for certain non-GSS scenarios. These variations form the basis of this specification.

1.4 Relationship to Other Protocols

SPNEGO requires at least one other GSS-compatible authentication protocol to be present for it to work. It does not depend on a specific protocol, although in practice, that protocol has typically been the Kerberos protocol.

Many application protocols make use of SPNEGO as their authentication protocol. Such protocols include the Common Internet File System (CIFS)/Server Message Block (SMB) (for more information, see [MS-SMB]); HTTP (for more information, see [HTTPAUTH]); Microsoft RPC (for more information, see [MS-RPCE]); and the Lightweight Directory Access Protocol (LDAP) (for more information, see [RFC2251]).

SPNEGO is a meta protocol that travels entirely in other application protocols; it is never used directly without an application protocol.

After SPNEGO has completed the ferrying of the other security protocol's authentication tokens, SPNEGO is finished. All further access to security context state and per-message services, such as signatures or encryption, is done by directly using the "real" security protocol whose authentication tokens were communicated via SPNEGO.

1.5 Prerequisites/Preconditions

Because SPNEGO relies on other security protocols that perform authentication, those protocols must be available to SPNEGO for it to operate. The set of protocols is implementation-dependent upon the installation.<1>

Applications typically establish a connection before they invoke SPNEGO, although establishing a connection before invoking SPNEGO is not required by the SPNEGO protocol.

1.6 Applicability Statement

As a GSS protocol, SPNEGO can be used almost anywhere that an application protocol uses GSS to perform authentication. The protocol must be connection-oriented because it is not designed to tolerate packet loss; datagram-only protocols cannot support negotiation of this form.

1.7 Versioning and Capability Negotiation

SPNEGO does not contain any versioning capacity. The same is true for capabilities: any capability negotiation must be performed by the actual authentication protocols that SPNEGO is carrying.

1.8 Vendor-Extensible Fields

Although SPNEGO itself is inherently vendor-extensible, the extensions that Microsoft has made to the protocol do not contain any extensible fields.

1.9 Standards Assignments

SPNEGO has been assigned the following **object identifier (OID)**:<2>

```
iso.org.dod.internet.security.mechanism.snego
(1.3.6.1.5.5.2)
```

2 Messages

The following sections specify the message syntax and transport for Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions.

2.1 Transport

SPNEGO is transported only when encapsulated in an application protocol. As such, it can travel over whatever transports the application protocol uses. By itself, SPNEGO never causes network traffic.

2.2 Message Syntax

The messages that the base SPNEGO protocol uses are specified in [RFC4178], in terms of ASN.1, as specified in [X680]. There are only two messages in SPNEGO, negTokenInit and negTokenResp, both of which are defined in [RFC4178].

The negTokenInit message is sent from the client to the server and is used to begin the negotiation. The client uses that message to specify the set of authentication mechanisms that are supported and an opportunistic authentication message from the mechanism that the client believes will be agreed upon with the server.

The negTokenResp message is used thereafter as the server selects the mechanism to use, and the two parties exchange authentication messages that are wrapped in the negTokenResp message until completion. Microsoft has extended standard SPNEGO by the addition of the NegTokenInit2 message.

2.2.1 NegTokenInit2 Extension

When the NegTokenInit2 message is used in the Windows extension, the message is structured as follows:<3>

```
NegHints ::= SEQUENCE {
        hintName[0] GeneralString OPTIONAL,
        hintAddress[1] OCTET STRING OPTIONAL
}
NegTokenInit2 ::= SEQUENCE {
        mechTypes[0] MechTypeList OPTIONAL,
        reqFlags [1] ContextFlags OPTIONAL,
        mechToken [2] OCTET STRING OPTIONAL,
        negHints [3] NegHints OPTIONAL,
        mechListMIC [4] OCTET STRING OPTIONAL,
        ...
}
```

mechTypes: The list of authentication mechanisms that are available, by OID, as specified in [RFC4178] section 4.1.

reqFlags: Never present. MUST be omitted by the sender. Note that the encoding rules, as specified in [X690], require that this structure not be present at all, not just be zero.

mechToken: Never present. MUST be omitted by the sender. Note that the encoding rules, as specified in [X690], require that this structure not be present at all, not just be zero.

negHints: The server supplies the negotiation hints using a **negHints** (negotiation hints) structure that is assembled as follows:

- **hintName**: The name of the server principal, which is the service principal on the server in the form user-name@domain-name. The name is expressed in ANSI encoding, which uses an **OEM code page** that the local system defines. For two parties to use this extension, the OEM code page must be agreed upon out-of-band of this protocol. Although typed as a GeneralString, the hintName does not actually allow the full range of encodings of GeneralString. For more information about how this field is populated, see section 3.2.5.1.
- **hintAddress**: Never present. MUST be omitted by the sender. Note that the encoding rules, as specified in [X690], require that this structure not be present at all, not just be zero.

mechListMIC: Never present. MUST be omitted by the sender. Note that the encoding rules, as specified in [X690], require that this structure not be present at all, not just be zero.

Note In the ASN.1 description above, the NegTokenInit2 message occupies the same context-specific message ID (0) as does NegTokenInit in SPNEGO.

2.2.2 User to User Session Extension

The presence of the following OID in the mechTypes list within a NegTokenInit or NegTokenInit2 message indicates a request for a user-to-user session:

{iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) krb5(2) usertouser(3)}

2.2.2.1 User to User Session Request Extension

The syntax of this list item is as defined in [UUKA-GSSAPI] section 4.

2.2.2.2 User to User Session Reply Extension

The syntax of this list item is as defined in [UUKA-GSSAPI] section 4.

3 Protocol Details

3.1 Common Details

The following are common variations, as specified in [RFC4178], for both client and server processing in the SPNEGO Protocol Extensions.

3.1.1 Abstract Data Model

SPNEGO exports a set of abstract parameters that describe the security services that a caller wants to have available for use on the communication session after it has been established. SPNEGO cannot directly act on these parameters because it does not perform the actual authentication. They are passed through to the underlying security protocols as an indication of the caller's eventual plans. These parameters are:

- Integrity: Indicates that the caller wants to sign messages so that they cannot be tampered with while in transit.
- Replay Detect: Indicates that the caller wants to sign messages so that they cannot be replayed.
- Sequence Detect: Indicates that the caller wants to sign messages so that they cannot be sent out of order.
- Confidentiality: Indicates that the caller wants to encrypt messages so that they cannot be read while in transit.
- Delegate: Indicates that the caller wants to make its own identity available to the server for further identification to other services.
- Mutual Authentication: Indicates that the client and server MUST authenticate each other; unidirectional authentication is not permissible.

  These flags correspond to the **reqFlags:ContextFlags** field in the NegTokenInit structure. As specified in [RFC4178], the **reqFlags:ContextFlags** field is now only for legacy purposes and SHOULD NOT be filled in. For more information about the **reqFlags:ContextFlags** field, see section 3.1.5.3.

Extended Error: Indicates that the caller wants the underlying protocol to perform the extended error handling, potentially including retries within the GSS exchange.

Additionally, the NegTokenInit message MAY be fragmented into several smaller byte structures within the length specified by the application protocol, via the parameter GSS_C_FRAGMENT_TO_FIT. A packet containing the ASN.1 header but an incomplete body is sent as a result of the fragmentation. By utilizing the length specified in the ASN.1 header of the initial packet, the Server determines the number of the bytes necessary to assemble the original NegTokenInit structure and acquires them by virtue of GSS_S_CONTINUE_NEEDED semantics. The subsequent packages each contain an ASN.1 header and the individual bytes structure composing the remaining NegTokenInit message. This allows the Server to assemble the original NegTokenInit and be agnostic to the Windows-specific fragmentation performed on the Client. The Server MUST return GSS_S_CONTINUE_NEEDED status with an empty output token in order to get the next byte structure from the client, and it MUST repeat this for all of the byte structures received. If the context is terminated (for example, because the network connection to the other entity is interrupted) before reassembly of the NegTokenInit message is complete, the entire NegTokenInit message MUST be discarded.

3.1.2 Timers

There are no timers.

3.1.3 Initialization

There are no changes to initialization.

3.1.4 Higher-Layer Trigger Events

There are no changes to higher-layer trigger events.

3.1.5 Message Processing Events and Sequencing Rules

The following fields are processed differently than as specified in [RFC4178].

3.1.5.1 mechListMIC Processing

[RFC2478] inadequately specifies the processing of the mechanism list Message Integrity Code, or **mechListMIC** field. [RFC4178] clarifies the processing of the **mechListMIC** field.<4>

3.1.5.2 mechTypes Identification of Kerberos

<5>

3.1.5.3 reqFlags Processing

[RFC2478], the predecessor to [RFC4178], includes the **reqFlags** field in the protocol. This field is intended for the client to indicate the requested behavior according to the GSS abstract variables, such as confidentiality and integrity. However, the **reqFlags** field is not covered by the signature of the message; therefore, it can be tampered with while in transit.

In the revisions that are specified in [RFC4178], use of this field is explicitly discouraged due to the lack of integrity protection. The acceptor (server) MUST ignore the **reqFlags**, if present.<6>

3.1.6 Timer Events

There are no timer events.

3.1.7 Other Local Events

There are no other local events.

3.2 Server (Acceptor) Role Details

3.2.1 Abstract Data Model

There are no changes to the data model.

3.2.2 Timers

There are no timers.

3.2.3 Initialization

There are no changes to initialization.

3.2.4 Higher-Layer Triggered Events

There are no changes to higher-layer triggered events.

3.2.5 Message Processing Events and Sequencing Rules

The following fields are processed differently than they are specified in [RFC4178].

3.2.5.1 NegTokenInit2 Variation for Server-Initiation

Standard GSS has a strict notion of client (initiator) and server (acceptor). If there is no authentication attempt in progress (an attempt is expressed as a security context in GSS-API terms), no token is expected from the server.

For historical reasons, Microsoft SPNEGO allows the server to generate a NegTokenInit2 message and send it to the client. Thus, the client logic for handling messages is extended to allow for a token to be received by the client without the client sending a token out first.

The server generates a NegTokenInit2 message by creating a **mechList** sequence that includes the OIDs of the security protocols that are present and available on the server. This message is encoded in the **mechList** field.

In the **negHints** field, the server places its own name, expressed as ANSI encoding, in the **hintName** field. For more information about how the **hintName** field is populated, see section 2.2.1.

The remaining fields MUST be omitted and not transmitted. The NegTokenInit2 token is then passed to the client within the application protocol.<7>

3.2.5.2 Universal Receiver

<8>

3.2.6 Timer Events

There are no timer events.

3.2.7 Other Local Events

There are no other local events.

3.3 Client (Initiator) Role Details

3.3.1 Abstract Data Model

There are no changes to the data model.

3.3.2 Timers

There are no timers.

3.3.3 Initialization

There are no changes to initialization.

3.3.4 Higher-Layer Triggered Events

There are no changes to higher-layer triggered events.

3.3.5 Message Processing Events and Sequencing Rules

The following field is processed differently than it is specified in [RFC4178].

3.3.5.1 NegTokenInit2 Variation for Server-Initiation

Standard GSS has a strict notion of client (initiator) and server (acceptor). If there is no authentication attempt (expressed as a security context in GSS-API terms) in progress, no token is expected from the server. As a performance optimization, SPNEGO Protocol Extensions can generate a NegTokenInit2 (section 2.2.1) message from the server and send it to the client. As such, the client logic for handling messages is extended to allow for a token to be received without the client being the initiator.<9>

All other fields MUST be ignored if they are present.

3.3.6 Timer Events

There are no timer events.

3.3.7 Other Local Events

There are no other local events.

4 Protocol Examples

The protocol variation is a single message addition, as specified in [RFC4178], so only a sample message can be shown. The following is the actual message from a server, in this case, a server that is named "yellowdog3.home.yellow-dogs.com", a member of a Windows Server 2003 domain that is named "home.yellow-dogs.com". Recall that SPNEGO messages are always carried in another application protocol; this information was extracted from the CIFS/SMB protocol. Specifically, this NegTokenInit2 (section 2.2.1) message was embedded in the SecurityBlob field of the SMB_COM_NEGOTIATE reply. For more information, see [MS-SMB] section 2.2.3.

```
00000000  60 66 06 06 2b 06 01 05-05 02 a0 5c 30 5a a0 30
          'f..+......\0Z.0
00000010  30 2e 06 09 2a 86 48 82-f7 12 01 02 02 06 09 2a
          0...*.H........*
00000020  86 48 86 f7 12 01 02 02-06 0a 2a 86 48 86 f7 12
          .H........*.H...
00000030  01 02 02 03 06 0a 2b 06-01 04 01 82 37 02 02 0a
          ......+.....7...
00000040  a3 26 30 24 a0 22 1b 20-79 65 6c 6c 6f 77 64 6f
          .&0$.". yellowdo
00000050  67 33 24 40 48 4f 4d 45-2e 59 45 4c 4c 4f 57 2d
          g3$@HOME.YELLOW-
00000060  44 4f 47 53 2e 43 4f 4d-
          DOGS.COM
```

In this packet, the server has sent a special NegTokenInit2 message. The text that follows a portion of the message explains that part of the message. The message, at byte offsets 0x02 through 0x09 inclusive, is:

```
[Application 0]
  Object ID(6) 1.3.6.1.5.5.2
```

Standard GSS, as specified in [RFC2743], predicates indicating application sequence with the OID of SPNEGO.

```
[Context Specific 0]
   [30] Sequence
```

NegTokenInit message, a constructed sequence, at byte offsets 0x10 through 0x3f inclusive, is:

```
[a0] [Context Specific 0]
   [30] Sequence
      [6] Object ID 1.2.840.48018.1.2.2
      [6] Object ID 1.2.840.113554.1.2.2
      [6] Object ID 1.2.840.113554.1.2.2.3
      [6] Object ID 1.3.6.1.4.1.311.2.2.10
```

Next is the mechList element, which is a list of OIDs. This list comprises the "old" Kerberos OID (as specified in section 3.1.5.2), the "real" Kerberos OID, the Kerberos user-to-user protocol variant that is exposed through GSS-API, and the OID that is assigned to NTLM.

```
[a3] [Context Specific 3]
   [30] Sequence
      [a0] [Context Specific 0]
         [1b] General String "yellowdog3$@HOME.YELLOW-DOGS.COM"
```

A general string that is the name of the service principal is encoded into the **negHint** field. In this case, the service principal name is the user name of yellowdog3$ and the domain name of HOME.YELLOW-DOGS.COM. Note that in this example, the service name is the machine principal name.

5 Security

The following sections specify the security considerations for implementers of the SPNEGO Protocol Extensions and an index of the security parameters.

5.1 Security Considerations for Implementers

Implementers of the SPNEGO Protocol Extensions should be aware of the correct use of the hint data that the server sends, as specified in section 3.3.5.1.

5.2 Index of Security Parameters

Security parameter	Section
GSS context parameters	NegTokenInit Variation for Server-Initiation (section 3.3.5.1)

6 Appendix A: Windows Behavior

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

- Windows Vista
- Windows Server 2003
- Windows XP
- Windows 2000

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 1.5: By default, Windows 2000 has available the Kerberos protocol and NTLM. The interface for authentication protocols in Windows 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.

<2> Section 1.9: Microsoft assigns an integer value to all security protocols. Windows implementations of SPNEGO automatically assign an object identifier (OID) to alternate security protocols that are installed on a system that runs Windows, and that do not have an official OID from the Internet Assigned Numbers Authority (IANA) or the International Organization for Standardization (ISO). This assignment is done by appending the integer value mentioned in section 1.9 to the Microsoft security protocol OID arc. Specifically, the OIDs that are generated fall under:

iso(1).org(3).dod(6).internet(1).private(4).enterprise(1).Microsoft(311).security(2).mechanisms(2).xxx

where *xxx* is the integer value that is assigned, as defined in section 1.9. Windows-based computers may negotiate mechanisms in addition to the default authentication mechanisms, if such a mechanism is installed on a specific Windows system.

<3> Section 2.2.1: The NegTokenInit2 message is generated in Windows Vista, Windows Server 2003, Windows XP, and Windows 2000.

<4> Section 3.1.5.1: Windows Server 2003, Windows XP, and Windows 2000 do not process the **mechListMIC** field. No **mechListMIC** value is produced when either the client or server completes the exchange. If a **mechListMIC** value is supplied to either the client or server, it is ignored. If the initiator in the GSS exchange has the last GSS token, the server returns a NegTokenResp token that has the **negState** field set to accept_complete and no **mechListMIC** field.

<5> Section 3.1.5.2: Windows versions offer and accept two distinct OIDs as identifiers for the Kerberos authentication mechanism. The SPNEGO extensions consider both OIDs as equivalent and make no distinction between them.

Windows 2000 incorrectly encoded the OID for the Kerberos protocol. Rather than the OID { iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) krb5(2) }, an implementation error truncated the values at 16 bits. Therefore, the OID became { iso(1) member-body(2) United States(840) ???(48018) infosys(1) gssapi(2) krb5 (2) }. 48018 is not a known member under that OID branch; this is an error.

Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP do not truncate the value, and they correctly offer and receive 1.2.840.113554.1.2.2. For compatibility reasons, the erroneous value (1.2.840.48018.1.2.2) is still offered and accepted by systems that run Windows 2000. The presence of this value can be used to determine that the peer is a Windows 2000 implementation.

<6> Section 3.1.5.3: Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, and Windows 2000 set this field to 0.

<7> Section 3.2.5.1: This behavior is present on Windows Server 2003, Windows XP, and Windows 2000. It is not present in Windows Vista. When encoding the name, the configured locale on the computer is used for the resulting character set. For example, on a computer that is configured to use the Japanese locale, the string is encoded by using ANSI Code, page 932. For more information about the code pages that are used by Windows, see [MSDN-CP].

<8> Section 3.2.5.2: This behavior is present on Windows Vista, Windows Server 2003, Windows XP, and Windows 2000. For backward compatibility and historical reasons, a Windows implementation of SPNEGO has the following behavior: it accepts raw Kerberos messages that are based on [RFC4121] and [RFC4120], and it accepts raw NTLM messages that are not embedded in [RFC4178] SPNEGO messages. This behavior is known as the universal receiver behavior.

<9> Section 3.3.5.1: This message is supported on Windows Server 2003, Windows XP, and Windows 2000.Windows Vista does not generate this message.

All versions of Windows that support this message also accept the mechList message from the server as described. All versions of Windows that support this message treat the server name as an optional indicator of the domain of the server only and use this message to select among possible sets of credentials. There is no guarantee that any Windows-based client will use this value in any discernable way. For example, if the client and server support the same set of security mechanisms, the client will continue to produce the same list as if the server had provided no list at all. All Windows versions ignore the server name in the NegHint structure by default; it was used as a development aid during Windows 2000 and is only used when an undocumented local configuration value is set. The hint support was retained in anticipation of cross-forest scenarios, but it was never used. Therefore, the removal of the hint in Windows Vista. The Windows implementation where this message is supported never alters the target name that is expressed by the application protocol above SPNEGO.

7 Index

T

U

V

W