NETWORK WORKING GROUP
Internet-Draft
Updates: 4178 (if approved)
Intended status: Standards Track
Expires: July 7, 2011

M. Short
L. Zhu
K. Damour
D. McPherson
Microsoft Corporation

January 3, 2011

SPNEGO Extended Negotiation (NEGOEX) Security Mechanism
draft-zhu-negoex-04

Abstract

This document defines the SPNEGO Extended Negotiation (NEGOEX) Security Mechanism. NEGOEX enhances the capabilities of SPNEGO by providing a security mechanism which can be negotiated by the SPNEGO protocol as defined in RFC4178.

The NEGOEX protocol itself is a security mechanism negotiated by SPNEGO. When the NEGOEX security mechanism is selected by SPNEGO, NEGOEX provides a method allowing selection of a common authentication protocol based on factors beyond just the fact that both client and server support a given security mechanism. NEGOEX OPTIONALLY adds a pair of meta-data messages for each negotiated security mechanism. The meta-data exchange allows security mechanisms to exchange auxiliary information such as trust configurations, thus NEGOEX provides more flexibility than just exchanging security mechanism OIDs in SPNEGO.

NEGOEX preserves the optimistic token semantics of SPNEGO and applies that recursively. Consequently a context establishment mechanism token can be included in the initial NEGOEX message, and NEGOEX does not require an extra round-trip when the initiator's optimistic token is accepted by the target.

Similar to SPNEGO, NEGOEX defines a few new GSS-API extensions that a security mechanism MUST support in order to be negotiated by NEGOEX. This document defines these GSS-API extensions.

Unlike SPNEGO however, NEGOEX defines its own way for signing the protocol messages in order to protect the protocol negotiation. The NEGOEX message signing or verification can occur before the security context for the negotiated real security mechanism is fully established.

Status of this Memo

This Internet-Draft is submitted in full conformance with the

Copyright Notice

Table of Contents

1. Introduction

If more than one GSS-API mechanism is shared between the initator and the acceptor, the Simple and Protected (GSS-API) Negotiation Mechanism (SPNEGO) as defined in [RFC4178] can be deployed to choose a mutually preferred one. This pseudo mechanism does well in the most basic scenarios but suffers from a couple of drawbacks, notably:

- o Since the SPNEGO negotiation is based on purely on exchanging security mechanism OIDs, security mechanisms can be selected which cannot successfully authenticate the initator. Just because an initator and acceptor support the same security mechanism does not mean that they have a mutually trusted authentication authority. In such cases, the authentication will fail with the preferred security mechanism, but might succeed with another common mechanism.

- o Secondly, the SPNEGO negotiation model is inadequate when the choice cannot be made by the acceptor in the initial response. In SPNEGO, the negotiation information is sent one-way from the initiator for the acceptor to make a choice, and the acceptor must choose one when it makes the initial response. This negotiation model is counter intuitive. The selection of a security mechanism is typically the result of selecting one type of credentials from the available set, and the initiator typically does not wish to reveal credentials information often associated with user identities. In practice, in order to operate in this model, the Kerberos GSS-API mechanism [RFC4121] must acquire the context establishment token in the initial call to GSS_Init_sec_context(). If the initiator fails to acquire the initial Kerberos GSS-API context token, it must not offer Kerberos; otherwise the SPNEGO context negotiation will fail without being able to select the next available mechanism that could work. Obtaining the initial Kerberos GSS-API context token may require multiple round-trips of network calls and the cost of the operation can be substantial. It is suboptimal when multiple GSS-API mechanisms have to add the extra cost that would not exist if the negotiated security mechanism were selected based on configuration.

The SPNEGO Extended Negotiation (NEGOEX) Security Mechanism is designed to address these concerns. NEGOEX is a security mechanism that is negotiated by SPNEGO, and when negotiated, it can recursively negotiate other security mechanisms.

Any security mechanism negotiated by NEGOEX MUST support integrity protection and addition GSS-API interfaces specified in Section 8.

The basic form of NEGOEX works as follows:

1. The initiator proposes a list of mechanisms in decreasing preference order. For each of these mechanism, NEGOEX OPTIONALLY includes a mechanism specific meta-data token. GSS-API extensions are defined later in this document for NEGOEX to query the meta-data token for inclusion in the NEGOEX message.

2. The acceptor then passes the meta-data token from the initiator to the intended security mechanism. A meta-data token for a security mechanism not supported on the acceptor side is ignored. New GSS-API extensions are defined later in this document for a security mechanism to consume the meta-data token. When processing the received meta-data tokens, a security mechanism that reports a failure is removed from the set of mutually supported mechanisms. The acceptor then responds with the list of mutually supported mechanisms in decreasing preference order. For each of these mechanism, NEGOEX again OPTIONALLY supplies a mechanism specific meta-data token in the response which it obtains from each remaining supported mechanism via the new GSS-API extensions described in the initial step.

3. The initiator then passes the meta-data tokens to the intended security mechanisms by invoking the new GSS-API extensions. When processing the received meta-data token, a security mechanism that reports a failure is removed from the set of mutually supported mechanisms for this negotiation context. The initiator then selects one from the set of mutually-supported mechanisms. If more than one security mechanism is available, unless otherwise specified, the highest one in the acceptor's preference order SHOULD be selected. Later when the common security mechanism is identified, the security mechanism may also negotiate mechanism-specific options during its context establishments. This will be inside the mechanism tokens, and invisible to the NEGOEX protocol during step 5.

4. The selected security mechanism provides keying materials to NEGOEX via new GSS-API extensions which defined later in this document. NEGOEX signs and verifies the negotiation NEGOEX messages to protect the negotiation.

5. The initiator and the acceptor proceed to exchange tokens until the GSS-API context for selected security mechanism is established. Once the security context is established, the per-message tokens are generated and verified in accordance with the selected security mechanism.

NEGOEX does not work outside of SPNEGO. When negotiated by SPNEGO, NEGOEX uses the concepts developed in the GSS-API specification [RFC2743]. The negotiation data is encapsulated in context-level

tokens. Therefore, callers of the GSS-API do not need to be aware of the existence of the negotiation tokens but only of the SPNEGO pseudo-security mechanism.

In its basic form NEGOEX requires at least one extra round-trip. Network connection setup is a critical performance characteristic of any network infrastructure and extra round trips over WAN links, packet radio networks, etc. really make a difference. In order to avoid such an extra round trip the initial security token of the preferred mechanism for the initiator may be embedded in the initial NEGOEX token. The optimistic mechanism token may be accompanied by the meta-data tokens and the optimistic mechanism token MUST be that of the first mechanism in the list of the mechanisms proposed by the initiator. The NEGOEX MESSAGE_TYPE_INITIATOR_NEGO message that contains signatures for protecting the NEGOEX negotiation may also accompany the optimistic mechanism token. If the target preferred mechanism matches the initiator's preferred mechanism, and when the NEGOEX negotiation protection messages are included along with the mechanism token, no additional round trips are incurred by using the NEGOEX protocol with SPNEGO.

NEGOEX does not update the ASN.1 structures of SPNEGO [RFC4178] because a widely deployed SPNEGO implementation does not have the ASN.1 extensibility marker in the message definition. There is no change to the SPNEGO messages.

NEGOEX uses a C-like definition language to describe message formats.

The rest of the document is organized as follows:

- Section 3 defines the encoding of NEGOEX data structures and all the primitive data types.
- Section 6 describes the cryptographic framework required by the NEGOEX for protecting the NEGOEX negotiation.
- Section 7 defines the NEGOEX messages and the NEGOEX protocol.
- Section 8 defines the new GSS-API extensions that a security mechanism MUST support in order to be negotiated by NEGOEX.
- Section 9 contains the security considerations for NEGOEX.
- Appendix A contains all the protocol constructs and constants.

2. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Presentation Language and Primitive Data Types

The following very basic and somewhat casually defined presentation syntax will be used in all NEGOEX messages. Although it resembles the programming language "C" in its syntax, it would be risky to draw too many parallels. The purpose of this presentation language is to document NEGOEX only; it has no general application beyond that particular goal.

This section also defines all the primitive data types. The semantics of the data types is explained in the next section.

3.1. Basic Block Size

The representation of all data items is explicitly specified. The basic data block size is one octet. Multiple octet data items are concatenations of octets, from left to right, from top to bottom Unless otherwise specific a multi-octet numeric is in little endian order with the least significant octet first.

3.2. Miscellaneous

Comments start with "//"' and continue until the end of the line.

3.3. Constants

Constants are denoted using "#define" followed by the symbolic name and then the constant value.

3.4. Numbers

UCHAR is the data type for a one-octet number.

ULONG is the data type for a 4-octet number encoded in little endian.

USHORT is the data type for a 2-octet number encoded in little endian.

ULONG64 is the data type for a 8-octet number encoded in little endian.

GUID is the data type for a 16-octet number encoded in little endian.

3.5. Enum Types

An enum type is the data type for a number with a small number of permissible values. An instance of an enum type is a 4-octet number encoded in little endian.

The definition of an enum type follows the simple "C" convention.

MESSAGE_TYPE is an enum type defined as follows:

```
enum
{
    MESSAGE_TYPE_INITIATOR_NEGO = 0,
    MESSAGE_TYPE_ACCEPTOR_NEGO,
    MESSAGE_TYPE_INITIATOR_META_DATA,
    MESSAGE_TYPE_ACCEPTOR_META_DATA,
    MESSAGE_TYPE_CHALLENGE,
        // an exchange message from the acceptor
    MESSAGE_TYPE_AP_REQUEST,
        // an exchange message from the initiator
    MESSAGE_TYPE_VERIFY,
    MESSAGE_TYPE_ALERT,
} MESSAGE_TYPE;
```

MESSAGE_TYPE_INITIATOR_NEGO has the value 0, and MESSAGE_TYPE_ALERT has the value 7.

3.6. Typedef Declarations

A typedef creates a synonym for the type. This is used to create more meaningful names for existing types.

The following two type synonyms are defined.

```
typedef GUID AUTH_SCHEME;
typedef GUID CONVERSATION_ID;
```

3.7. Array Types

Arrays are a data structure which holds multiple variables of the same data type consecutively and the number of elements is fixed. An array is declared using "C" convention. The following defines an array of 32 octets.

```
UCHAR Random[32];
```

3.8. Constructed Types

Structure types may be constructed from primitive types for convenience. Each specification declares a new, unique type. The syntax for definition is much like that of C.

```
struct {
T1 f1;
T2 f2;
...
Tn fn;
} T;
```

Structure definitions may be embedded.

The following types are defined as constructed types:

```
struct
{
ULONG ExtensionType; // negative extensions are critical
BYTE_VECTOR ExtensionValue;
} EXTENSION;
```

An extension has two fields. The ExtensionType field indicates how the extension data should be interpreted. The ExtensionValue field contains the extension data.

```
//
// schemes defined for the checksum in the VERIFY message
//

struct
{
ULONG cbHeaderLength;
ULONG ChecksumScheme;
ULONG ChecksumType; // in the case of RFC3961 scheme, this is
// the RFC3961 checksum type
BYTE_VECTOR ChecksumValue;
} CHECKSUM;
```

The CHECKSUM structure contains 4 fields. The cbHeaderLength length contains the length of the structure defintion in octets, and this field has a value of 20.

The ChecksumScheme field describes how checksum is computed and verified. Currently only one value is defined.

```
#define CHECKSUM_SCHEME_RFC3961 1
```

When the value of the ChecksumScheme field is 1 (CHECKSUM_SCHEME_RFC3961), the ChecksumValue field contains a sequence of octets computed according to [RFC3961] and the ChecksumType field contains the checksum type value defined according

to [RFC3961].

4. Vector Types

Vectors are a data structure which holds multiple variables of the same data type consecutively and the number of elements is not fixed. A vector contains a fixed length header followed by a variable length payload. The header of a vector structure contains the count of elements and the offset to the payload. In this document all the offset fields are relative to the beginning of the containing NEGOEX message. The size of each element is specified by the vector type definition.

The following vector types are defined.

```
struct
{
    ULONG ByteArrayOffset; // each element contains an octet/byte
    ULONG ByteArrayLength;
} BYTE_VECTOR;
```

BYTE_VECTOR encapsulates a variable length array of octets (or bytes) that are stored consecutively. Each element in is a byte (8 bits).

```
struct
{
    ULONG AuthSchemeArrayOffset;
         // each element contains an AUTH_SCHEME
    USHORT AuthSchemeCount;
} AUTH_SCHEME_VECTOR;
```

AUTH_SCHEME_VECTOR encapsulates a variable length array of AUTH_SCHEMEs that are stored consecutively. Each element is a structure of the type AUTH_SCHEME.

```
struct
{
    ULONG ExtensionArrayOffset;
        // each element contains an EXTENSION
    USHORT ExtensionCount;
} EXTENSION_VECTOR;
```

EXTENSION_VECTOR encapsulates a variable length array of EXTENSIONs that are stored consecutively. Each element is a structure of the type EXTENSION.

5. NEGOEX Messages

The following structure is the MESSAGE_HEADER:

```
struct
{
ULONG64 Signature; // contains MESSAGE_SIGNATURE
MESSAGE_TYPE MessageType;
ULONG SequenceNum; // the message sequence number of this,
// conversation, starting with 0 and sequentially
// incremented
ULONG cbHeaderLength; // the header length of this message,
// including the message specific header, excluding the
// payload
ULONG cbMessageLength; // the length of this message
CONVERSATION_ID ConversationId;
} MESSAGE_HEADER;
```

The following structure is the NEGO_MESSAGE:

```
struct
{
MESSAGE_HEADER Header;
// MESSAGE_TYPE_INITIATOR_NEGO for the initiator,
// MESSAGE_TYPE_ACCEPTOR_NEGO for the acceptor
UCHAR Random[32];
ULONG64 ProtocolVersion;
// version of the protocol, this contains 0
AUTH_SCHEME_VECTOR AuthSchemes;
EXTENSION_VECTOR Extensions;
} NEGO_MESSAGE;
```

The following structure is the EXCHANGE_MESSAGE:

```
struct
{
MESSAGE_HEADER Header;
// MESSAGE_TYPE_CHALLENGE for the acceptor,
// or MESSAGE_TYPE_AP_REQUEST for the initiator
// MESSAGE_TYPE_INITIATOR_META_DATA for
// the initiator metadata
// MESSAGE_TYPE_ACCEPTOR_META_DATA for
// the acceptor metadata
AUTH_SCHEME AuthScheme;
BYTE_VECTOR Exchange;
// contains the opaque handshake message for the
// authentication scheme
} EXCHANGE_MESSAGE;
```

6. Cryptographic Computations

The message signing and verification in NEGOEX is based on [RFC3961]. [RFC3961] is used here as a generic framework and this application is not Kerberos specific.

A security mechanism MUST support [RFC3961] in order to be negotiated by NEGOEX.

7. The NEGOEX Protocol

This section describes the NEGOEX protocol and it defines NEGOEX messages in the order that the messages can appear on the wire. The enum type MESSAGE_TYPE defined in Section 3.5 lists all NEGOEX message types. A GSS-API context token for NEGOEX consists of one or more NEGOEX messages. If there is more than one NEGOEX message, these messages are concatenated together. The smallest data unit for NEGOEX to compute the checksum for negotiation protection is s NEGOEX message. Note that NEGOEX is not a GSS-API mechanism itself and the initial NEGOEX context establishment token does not follow the mechanism-independent token format defined in Section 3.1 of [RFC2743].

The object identifier of the NEGOEX within SPNEGO is iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) microsoft (311) security(2) mechanisms(2) negoex(30).

7.1. High-level NEGOEX Message Flow

The following text art summarizes the protocol message flow:

```
        Initiator                                            Acceptor

        INITIATOR_NEGO
        +*INITIATOR_META_DATA
        *AP_REQUEST
                                   --------->
                                                        ACCEPTOR_NEGO
                                                  ACCEPTOR_META_DATA*+
                                   <---------              CHALLENGE*

                                       .
                                       .

        *AP_REQUEST                --------->
                                   <---------              CHALLENGE*

                                       .
                                       .
        *AP_REQUEST
        VERIFY                     --------->
                                                           CHALLENGE*
                                   <---------                  VERIFY
        * Indicates optional or situation-dependent messages that are
        not always sent.
        + Indicates there can be more than one instance.
```

7.2. NEGOEX Supported Security Mechanisms

NEGOEX maintains an ordered list of supported security mechanisms names to determine priority of security mechanisms. A security mechanism negotiable by NEGOEX is identified by a unique identifier of data type AUTH_SCHEME defined in Section 3.5. Supported security mechanisms are referenced by their corresponding authentication scheme IDs. The authentication scheme ID of a security mechanism is returned to NEGOEX by calling GSS_Query_mechanism_info() with the name of the security mechnism as defined in Section 8.3.

7.3. ConversationID

Both initiator and acceptor must keep protocol state in the form of a GUID, which will be referred to hereafter as the ConversationID.

7.4. Generation of the Initiator Initial Token

The GSS-API initiator makes the first call to GSS_Init_sec_context() with no input token, and the output token will be a NEGO_MESSAGE message with the MESSAGE_TYPE_INITIATOR_NEGO message followed by zero

or more EXCHANGE_MESSAGE messages containing meta-data tokens, followed by zero or one AP_REQUEST messages containing an optimistic initial context token.

The initiator generates a cryptographic strength random 16 byte value, stores it as the ConversationID, then sets the MESSAGE_HEADER header field with the same name to that value. The ConversationID in subsequent NEGOEX messages MUST remain the same. The initiator also fills the Random field using a secure random number generator. The initiator fills the AuthSchemes with available security mechanisms supported by the initiator in decreasing preference order.

The extensions field contains NEGOEX extensions for future extensibility. There are no extensions defined in this document. All negative extension types (the highest bit is set to 1) are critical. If the receiver does not understand a critical extension, the authentication attempt must be rejected.

The initiator can OPTIONALLY include a meta-data token, one for each available security mechanism.

A meta-data token is returned to NEGOEX for a security mechanism using GSS_Query_meta_data() extension as defined in Section 8.1. If a non-empty meta-data token is returned, then the meta-data token is encapsulated in an EXCHANGE message with the message type MESSAGE_TYPE_INITIATOR_META_DATA. On GSS_Query_meta_data call failure, NEGOEX SHOULD remove the security mechanism from the set of authentication schemes to be negotiated.

The AuthScheme field signifies the security mechanism for which the EXCHANGE message is targeted. If a security mechanism fails to produce the metadata token, it should be removed from the list of supported security mechanism for this negotiation context.

If there is more than one exchange message, the order in which the exchange message is included bears no significance. In other words, the exchange messages are in an unordered set. The NEGO_MESSAGE MAY be followed by a set of MESSAGE_TYPE_INITIATOR_META_DATA messages as described above, in which case all the NEGOEX messages concatenated are returned as a single output token.

The first mechanism in the initiator proposed list can OPTIONALLY include its initial context token in an AP_REQUEST message.

Both an AP_REQUEST(short for MESSAGE_TYPE_AP_REQUEST) message and a INITIATOR_META_DATA(short for MESSAGE_TYPE_INITIATOR_META_DATA) message are instances of the EXCHANGE_MESSAGE structure with different message type values. An AP_REQUEST message contains the

type MESSAGE_TYPE_AP_REQUEST while an INITIATOR_META_DATA message contains the type MESSAGE_TYPE_INITIATOR_META_DATA.

7.5. Receipt of the Initial Initiator Token and Generation of the Initial Acceptor Response

Upon receipt of the NEGO_MESSAGE from the initiator, the acceptor verifies the NEGO_MESSAGE to make sure it is well-formed. The acceptor extracts the ConversationID from the NEGO_MESSAGE and stores it as the ConversationID for the context handle. The acceptor then computes the list of authentication schemes that are mutually supported by examining the set of security mechanisms proposed by the initiator and the meta-data tokens from the initiator. The meta-data tokens are passed to the security mechanism via GSS_Exchange_meta_data() as defined in Section 8.2. On GSS_Exchange_meta_data call failure, NEGOEX SHOULD remove the security mechanism from the set of authentication schemes to be negotiated.

The acceptor MUST examine the NEGOEX extensions in the NEGO_MESSAGE. If there is an unknown critical extension, the authentication must be rejected.

The acceptor's output token is a NEGO_MESSAGE but with the the Header.MessageType set to MESSAGE_TYPE_ACCEPTOR_NEGO followed by zero or more EXCHANGE_MESSAGE containing meta-data tokens. The AuthSchemes field contains the list of mutually supported security mechanism in decreasing preference order of the acceptor. The acceptor does not need to honor the preference order proposed by the initiator when computing its preference list.

As with the initiator, the acceptor can OPTIONALLY include a meta-data token, one for each available security mechanism.

A meta-data token is obtained by NEGOEX for a security mechanism using GSS_Query_meta_data() extension as defined in Section 8.1. If a non-empty meta-data token is returned, then the meta-data token is encapsulated in an EXCHANGE message with the message type MESSAGE_TYPE_ACCEPTOR_META_DATA. For a given security mechanism if a meta-token is received from the initiator, GSS_Query_meta_data() MUST be invoked on the acceptor side for that security mechanism, and the output meta-data token, if present, MUST be included in the NEGOEX reply. On GSS_Query_meta_data call failure, NEGOEX SHOULD remove the security mechanism from the set of authentication schemes to be negotiated.

7.6. Receipt of the Acceptor Initial Response and Completion of Authentication after the Negotiation Phrase

Upon receipt of the initial response token from the acceptor, the application calls GSS_Init_sec_context with the response token. The initiator verifies the NEGOEX message received to make sure it is well-formed. The initiator ensures the correct context handle by verifying that the ConversationID of the context handle matches the conversation ID in the NEGOEX message received. The initiator then computes the list of authentication schemes that are mutually supported by examining the set of security mechanisms returned by the acceptor and the meta-data tokens from the acceptor The meta-data tokens are passed to the security mechanism via GSS_Exchange_meta_data() as defined in Section 8.2. On GSS_Exchange_meta_data call failure, NEGOEX SHOULD remove the security mechanism from the set of authentication schemes to be negotiated.

The initiator MUST examine the NEGOEX extensions in the NEGO_MESSAGE. If there is an unknown critical extension, the authentication must be rejected.

After the initial exchange of NEGO_MESSAGE messages, the initiator MUST choose the negotiated security mechanism. The negotiated security mechanism cannot be changed once it is selected.

The initiator and the acceptor can then proceed to exchange handshake messages by returning GSS_S_CONTINUE_NEEDED to the calling application as determined by the negotiated security mechanism until its authentication context is established. The context tokens of the negotiated security mechanism are encapsulated in an EXCHANGE_MESSAGE. If the context token is from the initiator, the EXCHANGE_MESSAGE message has the message type MESSAGE_TYPE_AP_REQUEST; otherwise, the message type is MESSAGE_TYPE_CHALLENGE.

7.7. Finalizing Negotiation

After the security mechanism has been selected, the initiator and acceptor can use GSS_Inquire_context to obtain the Negoex_Verify_key as defined in Section 8.4 to determine if there is a shared key for the VERIFY message. When there is a shared key established returned by GSS_Inquire_context as defined in Section 8.4, a VERIFY message is produced using the required checksum mechanism per RFC 3961 and included in the output token. The returned protocol key is used as the base key in the parlance of RFC3961 to sign all the NEGOEX messages in the negotiation context.

A VERIFY message is a VERIFY_MESSAGE structure. The AuthScheme field signifies from which security mechanism the protocol key was obtained. The checksum is computed based on RFC3961 and the key usage number is 23 for the message signed by the initiator, 25 otherwise. The checksum is performed over all the previous NEGOEX messages in the context negotiation.

```
struct
{
    MESSAGE_HEADER Header; // MESSAGE_TYPE_VERIFY
    AUTH_SCHEME AuthScheme;
    CHECKSUM Checksum;
         // contains the checksum of all the previously
         // exchanged messages in the order they were sent.
} VERIFY_MESSAGE;
```

Note that the VERIFY_MESSAGE message can be included before the security context for the negotiated security mechanism is fully established.

8. Supporting GSS-API Extensions

This section defined all the required GSS-API extensions required by NEGOEX which must be supported by security mechanisms usable with NEGOEX.

8.1. GSS_Query_meta_data

Inputs:

- o input_context_handle CONTEXT HANDLE
- o targ_name INTERNAL NAME, optional
- o deleg_req_flag BOOLEAN,
- o mutual_req_flag BOOLEAN,
- o replay_det_req_flag BOOLEAN,
- o sequence_req_flag BOOLEAN,
- o conf_req_flag BOOLEAN,
- o integ_req_flag BOOLEAN,

Outputs:

- o metadata OCTET STRING,
- o output_context_handle CONTEXT HANDLE

Return major_status codes:

- o GSS_S_COMPLETE indicates that the context referenced by the input_context_handle argument is valid, and that the output metadata value represents the security mechanism's provided metadata. A security mechanism may return empty metadata.
- o GSS_S_NO_CONTEXT indicates that no valid context was recognized for the input context_handle provided. Return values other than major_status and minor_status are undefined.
- o GSS_S_NO_CRED indicates that no metadata could be returned about the referenced credentials either because the input cred_handle was invalid or the caller lacks authorization to access the referenced credentials.
- o GSS_S_UNAVAILABLE indicates that the authentication security service does not support this operation.
- o GSS_S_FAILURE indicates that the requested operation failed for reasons unspecified at the GSS-API level. Return values other than major_status and minor_status are undefined.

GSS_Query_meta_data is used to retrieve a security mechanism's metadata.

8.2. GSS_Exchange_meta_data

Inputs:

- o input_context_handle CONTEXT HANDLE
- o cred_handle CREDENTIAL HANDLE, optional
- o targ_name INTERNAL NAME, optional
- o deleg_req_flag BOOLEAN,
- o mutual_req_flag BOOLEAN,
- o replay_det_req_flag BOOLEAN,
- o sequence_req_flag BOOLEAN,
- o conf_req_flag BOOLEAN,
- o integ_req_flag BOOLEAN,
- o metadata OCTET STRING,

Outputs:

- o output_context_handle CONTEXT HANDLE

Return major_status codes:

- o GSS_S_COMPLETE indicates that the metadata was provided to the security mechanism.
- o GSS_S_NO_CONTEXT indicates that no valid context was recognized for the input context_handle provided. Return values other than major_status and minor_status are undefined.

- o GSS_S_NO_CRED indicates that the metadata passed requested credentials not available via this credential handle.
- o GSS_S_UNAVAILABLE indicates that the security mechanism does not support this operation.
- o GSS_S_FAILURE indicates that the requested operation failed for reasons unspecified at the GSS-API level. Return values other than major_status and minor_status are undefined.

GSS_Exchange_meta_data is used to provide the metadata to each security mechanism.

8.3. GSS_Query_mechanism_info

Inputs:

- o SecMechName STRING,

Outputs:

- o AuthScheme AUTH_SCHEME

Return major_status codes:

- o GSS_S_COMPLETE indicates that the authentication scheme value represents the security mechanism's AUTH_SCHEME.
- o GSS_S_FAILURE indicates that the security mechanism does not support NEGOEX. Return values other than major_status and minor_status are undefined.

GSS_Query_mechanism_info returns a security mechanism's authentication scheme value.

8.4. GSS_Inquire_context

The following output is added to GSS_Inquire_context as defined in [RFC2743].

Outputs:

- o Negoex_Verify_key OCTET STRING

This new output is the key to be used by NEGOEX for the VERIFY message.

9. Security Considerations

Security mechanism SHOULD support providing VERIFY key material.

This ensures that VERIFY messages are generated to make NEGOEX safe from downgrade attacks.

10. Acknowledgements

TBD.

11. IANA Considerations

There is no action required for IANA.

12. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.

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

[RFC4178] Zhu, L., Leach, P., Jaganathan, K., and W. Ingersoll, "The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism", RFC 4178, October 2005.

Appendix A. Protocol Data Structures and Constant Values

This section compiles all the protocol data structures and constant values.

```
#define MESSAGE_SIGNATURE    0x535458454f47454ei64
    // "NEGOEXTS"
```

```
struct
{
    ULONG ByteArrayOffset; // each element contains a byte
    ULONG ByteArrayLength;
} BYTE_VECTOR;

struct
{
    ULONG AuthSchemeArrayOffset;
        // each element contains an AUTH_SCHEME
    USHORT AuthSchemeCount;
} AUTH_SCHEME_VECTOR;

struct
{
    ULONG ExtensionArrayOffset;
        // each element contains an EXTENSION
    USHORT ExtensionCount;
} EXTENSION_VECTOR;

struct
{
    ULONG ExtensionType; // negative extensions are critical
    BYTE_VECTOR ExtensionValue;
} EXTENSION;

//
// schemes defined for the checksum in the VERIFY message
//

#define CHECKSUM_SCHEME_RFC3961  1

struct
{
    ULONG cbHeaderLength;
    ULONG ChecksumScheme;
    ULONG ChecksumType; // in the case of RFC3961 scheme, this is
       // the RFC3961 checksum type
    BYTE_VECTOR ChecksumValue;
} CHECKSUM;

typedef GUID AUTH_SCHEME;
typedef GUID CONVERSATION_ID;

enum
{
    MESSAGE_TYPE_INITIATOR_NEGO = 0,
    MESSAGE_TYPE_ACCEPTOR_NEGO,
```

```
            MESSAGE_TYPE_INITIATOR_META_DATA,
            MESSAGE_TYPE_ACCEPTOR_META_DATA,
            MESSAGE_TYPE_CHALLENGE,
                // an exchange message from the acceptor
            MESSAGE_TYPE_AP_REQUEST,
                // an exchange message from the initiator
            MESSAGE_TYPE_VERIFY,
            MESSAGE_TYPE_ALERT,
        } MESSAGE_TYPE;

        struct
        {
            ULONG64 Signature; // contains MESSAGE_SIGNATURE
            MESSAGE_TYPE MessageType;
            ULONG SequenceNum; // the message sequence number of this,
                   // conversation, starting with 0 and sequentially
                   // incremented
            ULONG cbHeaderLength; // the header length of this message,
               // including the message specific header, excluding the
               // payload
            ULONG cbMessageLength; // the length of this message
            CONVERSATION_ID ConversationId;
        } MESSAGE_HEADER;

        struct
        {
            MESSAGE_HEADER Header;
                     // MESSAGE_TYPE_INITIATOR_NEGO for the initiator,
                     // MESSAGE_TYPE_ACCEPTOR_NEGO for the acceptor
            UCHAR Random[32];
            ULONG64 ProtocolVersion;
                    // version of the protocol, this contains 0
            AUTH_SCHEME_VECTOR AuthSchemes;
            EXTENSION_VECTOR Extensions;
        } NEGO_MESSAGE;

        struct
        {
            MESSAGE_HEADER Header;
                 // MESSAGE_TYPE_CHALLENGE for the acceptor,
                 // or MESSAGE_TYPE_AP_REQUEST for the initiator
                // MESSAGE_TYPE_INITiATOR_META_DATA for
                // the initiator metadata
                // MESSAGE_TYPE_ACCEPTOR_META_DATA for
                // the acceptor metadata
            AUTH_SCHEME AuthScheme;
            BYTE_VECTOR Exchange;
                // contains the opaque handshake message for the
```

```
                // authentication scheme
        } EXCHANGE_MESSAGE;

        struct
        {
            MESSAGE_HEADER Header; // MESSAGE_TYPE_VERIFY
            AUTH_SCHEME AuthScheme;
            CHECKSUM Checksum;
                 // contains the checksum of all the previously
                 // exchanged messages in the order they were sent.
        } VERIFY_MESSAGE;

        struct
        {
            ULONG AlertType;
            BYTE_VECTOR AlertValue;
        } ALERT;

        //
        // alert types
        //

        #define ALERT_TYPE_PULSE             1

        //
        // reason codes for the heartbeat message
        //

        #define ALERT_VERIFY_NO_KEY          1

        struct
        {
            ULONG cbHeaderLength;
            ULONG Reason;
        } ALERT_PULSE;

        struct
        {
            ULONG AlertArrayOffset; // the element is an ALERT
            USHORT AlertCount; // contains the number of alerts
        } ALERT_VECTOR;

        struct
        {
            MESSAGE_HEADER Header;
            AUTH_SCHEME AuthScheme;
            ULONG ErrorCode; // an NTSTATUS code
            ALERT_VECTOR Alerts;
```

```
       } ALERT_MESSAGE;
```

Authors' Addresses

Michiko Short
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

Email: michikos@microsoft.com

Larry Zhu
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

Email: lzhu@microsoft.com

Kevin Damour
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

Email: kdamour@microsoft.com

Dave McPherson
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

Email: davemm@microsoft.com