NETWORK WORKING GROUP
Internet-Draft
Obsoletes: 2478 (if approved)
Expires: June 15, 2005

L. Zhu
P. Leach
K. Jaganathan
Microsoft Corporation
W. Ingersoll
Sun Microsystems
December 15, 2004

The Simple and Protected GSS-API Negotiation Mechanism
draft-ietf-kitten-2478bis-04

Status of this Memo

Copyright Notice

Abstract

This document specifies a negotiation mechanism for the Generic Security Service Application Program Interface (GSS-API) which is described in RFC 2743.

GSS-API peers can use this negotiation mechanism to choose from a common set of security mechanisms.

If per-message integrity services are available on the established mechanism context, then the negotiation is protected against an attacker forcing the selection of a mechanism not desired by the peers.

This mechanism replaces RFC 2478 in order to fix defects in that specification and to describe how to interoperate with implementations of that specification commonly deployed on the Internet.

Table of Contents

1. Introduction

The GSS-API [RFC2743] provides a generic interface which can be layered atop different security mechanisms such that if communicating peers acquire GSS-API credentials for the same security mechanism, then a security context may be established between them (subject to policy). However, GSS-API does not prescribe the method by which GSS-API peers can establish whether they have a common security mechanism.

The Simple and Protected GSS-API Negotiation (SPNEGO) mechanism defined here is a pseudo security mechanism, represented by the Object Identifier iso.org.dod.internet.security.mechanism.snego (1.3.6.1.5.5.2), which enables GSS-API peers to determine in-band whether their credentials support a common set of one or more GSS-API security mechanisms, and if so, to invoke the normal security context establishment for a selected common security mechanism. This is most useful for applications which depend on GSS-API implementations and share multiple mechanisms between the peers.

The SPNEGO mechanism negotiation is based on the following model: the initiator proposes a list of security mechanism(s), in decreasing preference order (favorite choice first), the acceptor (also known as the target) either accepts the initiator's preferred security mechanism (the first in the list), or chooses one that is available from the offered list, or rejects the proposed value(s). The target then informs the initiator of its choice.

Once a common security mechanism is chosen, mechanism-specific options MAY be negotiated as part of the selected mechanism's context establishment. These negotiations (if any) are internal to the mechanism and opaque to the SPNEGO protocol. As such they are outside the scope of this document.

If per-message integrity services are available on the established mechanism security context, then the negotiation is protected to ensure that the mechanism list has not been modified. In cases where an attacker could have materially influenced the negotiation, peers exchange message integrity code (MIC) tokens to confirm the mechanism list has not been modified. If no action of an attacker could have materially modified the outcome of the negotiation, the exchange of MIC tokens is optional (see Section 5). Allowing MIC tokens to be optional in this case provides interoperability with existing implementations while still protecting the negotiation. This interoperability comes at the cost of increased complexity.

In order to avoid an extra round trip, the first context establishment token of the initiator's preferred mechanism SHOULD be

embedded in the initial negotiation message (as defined in Section 4.2). (This mechanism token is referred to as the optimistic mechanism token in this document.) In addition, using the optimistic mechanism token allows the initiator to recover from non-fatal errors encountered trying to produce the first mechanism token before a mechanism can be selected. Implementations MAY omit the optimistic mechanism token in cases where the likelihood of the initiator's preferred mechanism not being selected by the acceptor is significant given the cost of generating it.

SPNEGO relies on 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 new pseudo-security mechanism. A failure in the negotiation phase causes a major status code to be returned: GSS_S_BAD_MECH.

2. Conventions Used in This Document

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. Negotiation Protocol

When the established mechanism context provides integrity protection, the mechanism negotiation can be protected. When acquiring negotiated security mechanism tokens, per-message integrity services are always requested by the SPNEGO mechanism.

When the established mechanism context supports per-message integrity services, SPNEGO guarantees that the selected mechanism is mutually preferred.

This section describes the negotiation process of this protocol.

3.1 Negotiation Description

The first negotiation token sent by the initiator contains an ordered list of mechanisms in decreasing preference order (favorite mechanism first), and optionally the initial mechanism token for the preferred mechanism of the initiator (i.e., the first in the list). (Note that the list MUST NOT contain mechanisms for which the client does not have appropriate credentials.)

The target then processes the token from the initiator. This will result in one of four possible states (as defined in Section 4.2.2) being returned in the reply message: accept_completed, accept_incomplete, reject, or request_mic. A reject state will terminate the negotiation; an accept_completed state indicates that not only was the initiator-selected mechanism acceptable to the target, but also that the optimistic mechanism token was sufficient to complete the authentication; an accept_incomplete state indicates that further message exchange is needed but the MIC token exchange as described in Section 5 is OPTIONAL; a request_mic state (this state can only be present in the first reply message from the target) indicates the MIC token exchange is REQUIRED if per-message integrity services are available.

Unless the preference order is specified by the application, the policy by which the target chooses a mechanism is an implementation-specific local matter. In the absence of an application specified preference order or other policy, the target SHALL choose the first mechanism in the initiator proposed list for which it has valid credentials.

In case of a successful negotiation, the security mechanism in the first reply message represents the value suitable for the target, chosen from the list offered by the initiator.

In case of an unsuccessful negotiation, the reject state is returned

and it is OPTIONAL to emit a context level negotiation token.

Once a mechanism has been selected, context establishment tokens specific to the selected mechanism are carried within the negotiation tokens.

Lastly, MIC tokens may be exchanged to ensure the authenticity of the mechanism list received by the target.

To avoid conflicts with the use of MIC tokens by SPNEGO, partially-established contexts MUST NOT be used for per-message calls. To guarantee this, the prot_ready_state [RFC2743] MUST be set to false on return from GSS_Init_sec_context() and GSS_Accept_sec_context() even if the underlying mechanism returned true.

3.2 Negotiation Procedure

The basic form of the procedure assumes that per-message integrity services are available on the established mechanism context, and it is summarized as follows:

(a) The GSS-API initiator invokes GSS_Init_sec_context() as normal, but requests that SPNEGO be used. SPNEGO can either be explicity requested or accepted as the default mechanism.

(b) The initiator GSS-API implementation emits a negotiation token containing a list of one or more security mechanisms that are available based on the credentials used for this context establishment, and optionally the initial mechanism token for the first mechanism in the list.

(c) The GSS-API initiator application sends the token to the target application. The GSS-API target application deposits the token by invoking GSS_Accept_sec_context(). The acceptor will do one of the following:

(I) If none of the proposed mechanisms are acceptable, the negotiation SHALL be terminated. GSS_Accept_sec_context indicates GSS_S_BAD_MECH. The acceptor MAY output a negotiation token containing a reject state.

(II) If either the initiator's preferred mechanism is not accepted by the target or this mechanism is accepted but it is not the acceptor's most preferred mechanism (i.e., the MIC token exchange as described in Section 5 is required), GSS_Accept_sec_context() indicates GSS_S_CONTINUE_NEEDED.

The acceptor MUST output a negotiation token containing a request_mic state.

(III) Otherwise if at least one additional negotiation token from the initiator is needed to establish this context, GSS_Accept_sec_context() indicates GSS_S_CONTINUE_NEEDED and outputs a negotiation token containing an accept_incomplete state.

(IV) Otherwise no additional negotiation token from the initiator is needed to establish this context, GSS_Accept_sec_context() indicates GSS_S_COMPLETE and outputs a negotiation token containing an accept_complete state.

If the initiator's preferred mechanism is accepted, and an optimistic mechanism token was included, this mechanism token MUST be deposited to the selected mechanism by invoking GSS_Accept_sec_context() and if a response mechanism token is emitted, it MUST be included in the response negotiation token. Otherwise, the target will not emit a response mechanism token in the first reply.

(d) The GSS-API target application returns the negotiation token to the initiator application. The GSS-API initiator application deposits the token by invoking GSS_Init_sec_context(). The security context initialization is then continued according to the standard GSS-API conventions for the selected mechanism, where the tokens of the selected mechanism are encapsulated in negotiation messages (see Section 4) until the GSS_S_COMPLETE is returned for both the initiator and the target by the selected security mechanism.

(e) MIC tokens are then either skipped or exchanged according to Section 5.

Note that the *_req_flag input parameters for context establishment are relative to the selected mechanism, as are the *_state output parameters. i.e., these parameters are not applicable to the negotiation process per se.

On receipt of a negotiation token on the target side, a GSS-API implementation that does not support negotiation would indicate the GSS_S_BAD_MECH status as if a particular basic security mechanism had been requested and was not supported.

When a GSS-API credential is acquired for the SPNEGO mechanism the implementation SHOULD produce a credential element for the SPNEGO

mechanism which internally contains GSS-API credential elements for all mechanisms for which the principal has credentials available, except for any mechanisms which are not to be negotiated, either as per implementation-, site- or application-specific policy. See Appendix A for interfaces for expressing application policy.

4. Token Definitions

The type definitions in this section assume an ASN.1 module definition of the following form:

```
SPNEGOASNOneSpec {
    iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanism(5) snego (2) modules(4) spec2(2)
} DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- rest of definitions here

END
```

This specifies that the tagging context for the module will be explicit and non-automatic.

The encoding of SPNEGO protocol messages shall obey the Distinguished Encoding Rules (DER) of ASN.1 as described in [X690].

4.1 Mechanism Types

In this negotiation model, each OID represents one GSS-API mechanism or one variant (see Section 6) of it according to [RFC2743].

```
MechType ::= OBJECT IDENTIFIER
    -- OID represents each security mechanism as suggested by
    -- [RFC2743]

MechTypeList ::= SEQUENCE OF MechType
```

4.2 Negotiation Tokens

The syntax of the initial negotiation tokens follows the initialContextToken syntax defined in Section 3.1 of [RFC2743]. The SPNEGO pseudo mechanism is identified by the Object Identifier specified in Section 1. Subsequent tokens MUST NOT be encapsulated in this GSS-API generic token framing.

This section specifies the syntax of the inner token for the initial message and the syntax of subsequent context establishment tokens.

```
NegotiationToken ::= CHOICE {
    negTokenInit    [0] NegTokenInit,
```

```
        negTokenResp    [1] negTokenResp
    }
```

4.2.1 negTokenInit

```
    NegTokenInit ::= SEQUENCE {
        mechTypes       [0] MechTypeList,
        reqFlags        [1] ContextFlags  OPTIONAL,
          -- maintained from RFC 2478 for backward compatibility,
          -- RECOMMENDED to be left out
        mechToken       [2] OCTET STRING  OPTIONAL,
        mechListMIC     [3] OCTET STRING  OPTIONAL,
        ...
    }
    ContextFlags ::= BIT STRING {
        delegFlag       (0),
        mutualFlag      (1),
        replayFlag      (2),
        sequenceFlag    (3),
        anonFlag        (4),
        confFlag        (5),
        integFlag       (6)
    }
```

This is the syntax for the inner token of the initial negotiation message.

mechTypes

This field contains one or more security mechanisms available for the initiator in decreasing preference order (favorite choice first).

reqFlags

This field, if present, contains the service options that are requested to establish the context. The context flags SHOULD be filled in from the req_flags parameter of GSS_Init_sec_context(). This field SHALL NOT have impact on the negotiation.

mechToken

This field, if present, contains the optimistic mechanism token.

mechlistMIC

This field, if present, contains a MIC token for the mechanism list in the initial negotiation message. This MIC token is computed according to Section 5.

4.2.2 negTokenResp

```
NegTokenResp ::= SEQUENCE {
    negState       [0] ENUMERATED {
        accept_completed    (0),
        accept_incomplete   (1),
        reject              (2),
        request_mic         (3)
    }                                  OPTIONAL,
      -- REQUIRED in the first reply from the target
    supportedMech   [1] MechType      OPTIONAL,
      -- present only in the first reply from the target
    responseToken   [2] OCTET STRING  OPTIONAL,
    mechListMIC     [3] OCTET STRING  OPTIONAL,
    ...
}
```

This is the syntax for all subsequent negotiation messages.

negState

This field, if present, contains the state of the negotiation. This can be:

accept_completed

No further negotiation message from the peer is expected, and the security context is established for the sender.

accept_incomplete

At least one more negotiation message from the peer is needed to establish the security context.

reject

The sender terminates the negotiation.

request_mic

The sender indicates that the exchange of MIC tokens, as described in Section 5, will be REQUIRED if per-message integrity services are available on the mechanism context to be established. This value SHALL only be present in the first reply from the target.

This field is REQUIRED in the first reply from the target, and it is OPTIONAL thereafter. When negState is absent the actual state should be inferred from the state of the negotiated mechanism context.

supportedMech

This field SHALL only be present in the first reply from the target. It MUST be one of the mechanism(s) offered by the initiator.

ResponseToken

This field, if present, contains tokens specific to the mechanism selected.

mechlistMIC

This field, if present, contains a MIC token for the mechanism list in the initial negotiation message. This MIC token is computed according to Section 5.

5. Processing of mechListMIC

If the mechanism selected by the negotiation does not support integrity protection, then no mechlistMIC token is used.

Otherwise, if the accepted mechanism is the most preferred mechanism of both the initiator and the acceptor, then the MIC token exchange, as described later in this section, is OPTIONAL. A mechanism is the acceptor's most preferred mechanism if there is no other mechanism which, had it been present in the mechanism list, the acceptor would have preferred over the accepted mechanism.

In all other cases, MIC tokens MUST be exchanged after the mechanism context is fully established.

a) The mechlistMIC token (or simply the MIC token) is computed over the mechanism list in the initial negotiation message by invoking GSS_GetMIC() as follows: the input context_handle is the established mechanism context, the input qop_req is 0, and the input message is the DER encoding of the value of type MechTypeList which is contained in the "mechTypes" field of the NegTokenInit. The input message is NOT the DER encoding of the type "[0] MechTypeList".

b) If the selected mechanism exchanges an even number of mechanism tokens (i.e., the acceptor sends the last mechanism token), the acceptor does the following when emitting the negotiation message containing the last mechanism token: if the MIC token exchange is optional, GSS_Accept_sec_context() either indicates GSS_S_COMPLETE and does not include a mechlistMIC token, or indicates GSS_S_CONTINUE_NEEDED and includes a mechlistMIC token and an accept_incomplete state; if the MIC token exchange is required, GSS_Accept_sec_context() indicates GSS_S_CONTINUE_NEEDED, and includes a mechlistMIC token. Acceptors that wish to be compatible with legacy Windows SPNEGO implementations as described in Appendix B should not generate a mechlistMIC token when the MIC token exchange is not required. The initiator then processes the last mechanism token, and does one of the following:

   (I) If a mechlistMIC token was included, and is correctly verified, GSS_Init_sec_context() indicates GSS_S_COMPLETE. The output negotiation message contains a mechlistMIC token, and an accept_complete state. The acceptor MUST then verify this mechlistMIC token.

(II) If a mechlistMIC token was included but is incorrect, the negotiation SHALL be terminated. GSS_Init_sec_context() indicates GSS_S_DEFECTIVE_TOKEN.

(III) If no mechlistMIC token was included, and the MIC token exchange is not required, GSS_Init_sec_context() indicates GSS_S_COMPLETE with no output token.

(IV) If no mechlistMIC token was included, but the MIC token exchange is required, the negotiation SHALL be terminated. GSS_Accept_sec_context() indicates GSS_S_DEFECTIVE_TOKEN.

c) In the case that the chosen mechanism exchanges an odd number of mechanism tokens (i.e., the initiator sends the last mechanism token), the initiator does the following when emitting the negotiation message containing the last mechanism token: if the negState was request_mic in the first reply from the target, a mechlistMIC token MUST be included, otherwise the mechlistMIC token is OPTIONAL. (Note that the MIC token exchange is required if a mechanism other than the initiator's first choice is chosen.) In the case that the optimistic mechanism token is the only mechanism token for the initiator's preferred mechanism, the mechlistMIC token is OPTIONAL. Whether or not the mechlistMIC token is included, GSS_Init_sec_context() indicates GSS_S_CONTINUE_NEEDED. Initiators that wish to be compatible with legacy Windows SPNEGO implementations as described in Appendix B should not generate a mechlistMIC token when the MIC token exchange is not required. The acceptor then processes the last mechanism token and does one of the following:

(I) If a mechlistMIC token was included and is correctly verified, GSS_Accept_sec_context() indicates GSS_S_COMPLETE. The output negotiation message contains a mechlistMIC token and an accept_complete state. The initiator MUST then verify this mechlistMIC token.

(II) If a mechlistMIC token was included but is incorrect, the negotiation SHALL be terminated. GSS_Accept_sec_context() indicates GSS_S_DEFECTIVE_TOKEN.

(III) If no mechlistMIC token was included but the mechlistMIC token exchange is not required, GSS_Accept_sec_context() indicates GSS_S_COMPLETE. The output negotiation message contains an accept_complete state.

(IV) In the case that the optimistic mechanism token is also the last mechanism token (when the initiator's preferred mechanism is accepted by the target) and the target sends a request_mic state but the initiator did not send a mechlistMIC token, the target then MUST include a mechlistMIC token in that first reply. GSS_Accept_sec_context() indicates GSS_S_CONTINUE_NEEDED. The initiator MUST verify the received mechlistMIC token and generate a mechlistMIC token to send back to the target. The target SHALL in turn verify the returned mechlistMIC token and complete the negotiation.

(V) If no mechlistMIC token was included and the acceptor sent a request_mic state in the first reply message (the exchange of MIC tokens is required), the negotiation SHALL be terminated. GSS_Accept_sec_context() indicates GSS_S_DEFECTIVE_TOKEN.

6. Extensibility

Two mechanisms are provided for extensibility. First, the ASN.1 structures in this specification MAY be expanded by IETF standards action. Implementations receiving unknown fields MUST ignore these fields.

Secondly, OIDs corresponding to a desired mechanism attribute (i.e., mechanism variants) may be included in the set of preferred mechanisms by an initiator. The acceptor can choose to honor this request by preferring mechanisms that have the included attributes. Future work within the Kitten working group is expected to standardize common attributes that SPNEGO mechanisms may wish to support. At this time it is sufficient to say that initiators MAY include OIDs that do not correspond to mechanisms. Such OIDs MAY influence the acceptor's choice of mechanism. As discussed in Section 5, if there are mechanisms that if present in the initiator's list of mechanisms might be preferred by the acceptor to the initiator's preferred mechanism, the acceptor MUST demand the MIC token exchange. As a consequence, acceptors MUST demand the MIC token exchange if they support negotiation of attributes not available in the initiator's preferred mechanism regardless of whether the initiator actually requested these attributes.

7. Security Considerations

In order to produce the MIC token for the mechanism list, the mechanism must provide integrity protection. When the selected mechanism does not support integrity protection, the negotiation is vulnerable: an active attacker can force it to use a security mechanism that is not mutually preferred but is acceptable to the target.

This protocol provides the following guarantees when per-message integrity services are available on the established mechanism context and the mechanism list was altered by an adversary such that a mechanism which is not mutually preferred could be selected:

a) If the last mechanism token is sent by the initiator, both peers shall fail;
b) If the last mechanism token is sent by the acceptor, the acceptor shall not complete and the initiator at worst shall complete with its preferred mechanism being selected.

The negotiation may not be terminated if an alteration was made but it had no material impact.

The protection of the negotiation depends on the strength of the integrity protection. In particular, the strength of SPNEGO is no stronger than the integrity protection of the weakest mechanism acceptable to GSS-API peers.

In all cases, the communicating peers are exposed to the denial of service threat.

8. IANA Considerations

This document has no actions for IANA.

9. Acknowledgments

The authors wish to thank Sam Hartman, Nicolas Williams, Ken Raeburn, Jeff Altman, Tom Yu, Cristian Ilac and Martin Rex for their comments and suggestions during development of this document.

Luke Howard provided a prototype of this protocol in Heimdal and resolved several issues in the initial draft.

Eric Baize and Denis Pinkas wrote the original SPNEGO specification [RFC2478] of which some of the text has been retained in this document.

10. References

10.1 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.

[X690] ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), ITU-T Recommendation X.690 (1997) | ISO/IEC International Standard 8825-1:1998.

10.2 Informative References

[RFC2478] Baize, E. and D. Pinkas, "The Simple and Protected GSS-API Negotiation Mechanism", RFC 2478, December 1998.

Authors' Addresses

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

EMail: lzhu@microsoft.com

Paul Leach
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

EMail: paulle@microsoft.com

Karthik Jaganathan
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

EMail: karthikj@microsoft.com

Wyllys Ingersoll
Sun Microsystems
1775 Wiehle Avenue, 2nd Floor
Reston, VA 20190
US

EMail: wyllys.ingersoll@sun.com

Appendix A. GSS-API Negotiation Support API

In order to provide to a GSS-API caller (either the initiator or the target or both) the ability to choose among the set of supported mechanisms a reduced set of mechanisms for negotiation, two additional APIs are defined:

- GSS_Get_neg_mechs() indicates the set of security mechanisms available on the local system to the caller for negotiation, for which appropriate credentials are available.
- GSS_Set_neg_mechs() specifies the set of security mechanisms to be used on the local system by the caller for negotiation, for the given credentials.

A.1 GSS_Set_neg_mechs call

Inputs:

- cred_handle CREDENTIAL HANDLE, -- NULL specifies default -- credentials
- mech_set SET OF OBJECT IDENTIFIER

Outputs:

- major_status INTEGER,
- minor_status INTEGER

Return major_status codes:

- GSS_S_COMPLETE indicates that the set of security mechanisms available for negotiation has been set to mech_set.
- GSS_S_FAILURE indicates that the requested operation could not be performed for reasons unspecified at the GSS-API level.

Allows callers to specify the set of security mechanisms that may be negotiated with the credential identified by cred_handle. This call is intended for support of specialized callers who need to restrict the set of negotiable security mechanisms from the set of all security mechanisms available to the caller (based on available credentials). Note that if more than one mechanism is specified in mech_set, the order in which those mechanisms are specified implies a relative preference.

A.2 GSS_Get_neg_mechs call

Input:

- cred_handle CREDENTIAL HANDLE -- NULL specifies default -- credentials

Outputs:

- major_status INTEGER,
- minor_status INTEGER,
- mech_set SET OF OBJECT IDENTIFIER

Return major_status codes:

- GSS_S_COMPLETE indicates that the set of security mechanisms available for negotiation has been returned in mech_set.
- GSS_S_FAILURE indicates that the requested operation could not be performed for reasons unspecified at the GSS-API level.

Allows callers to determine the set of security mechanisms available for negotiation with the credential identified by cred_handle. This call is intended for support of specialized callers who need to reduce the set of negotiable security mechanisms from the set of supported security mechanisms available to the caller (based on available credentials).

Note: The GSS_Indicate_mechs() function indicates the full set of mechanism types available on the local system. Since this call has no input parameter, the returned set is not necessarily available for all credentials.

Appendix B. Changes since RFC2478

SPNEGO implementations in Windows 2000/Windows XP/Windows Server 2003 have the following behavior: no mechlistMIC is produced and mechlistMIC is not processed if one is provided; if the initiator sends the last mechanism token, the acceptor will send back a negotiation token with an accept_complete state and no mechlistMIC token. In addition, an incorrect OID (1.2.840.48018.1.2.2) can be used to identify the GSS-API Kerberos Version 5 mechanism.

The following changes have been made to be compatible with these legacy implementations.

* NegTokenTarg is changed to negTokenResp and it is the message format for all subsequent negotiation tokens.
* NegTokenInit is the message for the initial negotiation message and that message only.
* mechTypes in negTokenInit is not optional.
* If the selected mechanism is also the most preferred mechanism for both peers, it is safe to omit the MIC tokens.

If at least one of the two peers implements the updated pseudo mechanism in this document, the negotiation is protected.

The following changes are to address the problems in RFC 2478.

* reqFlags is not protected therefore it should not impact the negotiation.
* DER encoding is required.
* GSS_GetMIC() input is clarified.
* Per-message integrity services are requested for the negotiated mechanism.
* Two MIC tokens are exchanged, one in each direction.

An implementation that conforms to this specification will not interoperate with a strict 2748 implementation. Even if the new implementation always sends a mechlistMIC token, it will still fail to interoperate. If it is a server, it will fail because it requests a mechlistMIC token using an option that older implementations simply do not support. Clients will tend to fail as well.

As an alternative to the approach chosen in this specification, we could have documented a correct behavior that is fully backward compatible with RFC 2478 and included an appendix on how to interoperate with existing incorrect implementations of RFC 2478.

As a practical matter, the SPNEGO implementers within the IETF have valued interoperability with the Microsoft implementations. We were

unable to choose to maintain reasonable security guarantees, maintain interoperability with the Microsoft implementations and maintain interoperability with correct implementations of RFC 2478. The working group was not aware of any RFC 2478 implementations deployed on the Internet. Even if there are such implementations, it is unlikely that they will interoperate because of a critical flaw in the description of the encoding of the mechanism list in RFC 2478.

With the approach taken in this specification, security is ensured between new implementations all the time while maintaining interoperability with the implementations deployed within the IETF community. The working group believes that this justifies breaking compatibility with a correct implementation of RFC 2478.

Appendix C. mechListMIC Computation Example

The following is an example to illustrate how the mechListMIC field would be computed.

The initial part of the DER encoding of NegTokenInit is constructed as follows (the "nn" are length encodings, possibly longer than one octet):

```
30 -- identifier octet for constructed SEQUENCE (NegTokenInit)
nn -- length

   -- contents octets of the SEQUENCE begin with
   -- DER encoding of "[0] MechTypeList":
   A0 -- identifier octet for constructed [0]
   nn -- length

      -- contents of the constructed [0] are DER encoding
      -- of MechTypeList (which is a SEQUENCE):
      30 -- identifier octet for constructed SEQUENCE
      nn -- length

         -- contents octets of the SEQUENCE begin with
         -- DER encoding of OBJECT IDENTIFIER:
         06 -- identifier octet for primitive OBJECT IDENTIFIER
         09 -- length
         2A 86 48 86 F7 12 01 02 02 -- Kerberos V5
                                    -- {1 2 840 113554 1 2 2}
```

If a mechlistMIC needs to be generated (according to the rules in Section 5), it is computed by using the DER encoding of the type MechTypeList data from the initiator's NegTokenInit token as input to the GSS_GetMIC() function. In this case, the MIC would be computed over the following octets:

```
DER encoding of MechTypeList:
30 nn 06 09 2A 86 48 86 F7 12 01 02 02 ...
```

Note that the identifier octet and lengh octet(s) for constructed [0] (A0 nn) are not included in the MIC computation.

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.