Internet Engineering Task Force (IETF)
Request for Comments: 5801
Category: Standards Track
ISSN: 2070-1721

S. Josefsson
SJD AB
N. Williams
Oracle

July 2010

Using Generic Security Service Application Program Interface (GSS-API) Mechanisms in Simple Authentication and Security Layer (SASL): The GS2 Mechanism Family

Abstract

This document describes how to use a Generic Security Service Application Program Interface (GSS-API) mechanism in the Simple Authentication and Security Layer (SASL) framework. This is done by defining a new SASL mechanism family, called GS2. This mechanism family offers a number of improvements over the previous "SASL/GSSAPI" mechanism: it is more general, uses fewer messages for the authentication phase in some cases, and supports negotiable use of channel binding. Only GSS-API mechanisms that support channel binding and mutual authentication are supported.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5801.

Table of Contents

1. Introduction

Generic Security Service Application Program Interface (GSS-API) [RFC2743] is a framework that provides security services to applications using a variety of authentication mechanisms. Simple Authentication and Security Layer (SASL) [RFC4422] is a framework to provide authentication and security layers for connection-based protocols, also using a variety of mechanisms. This document describes how to use a GSS-API mechanism as though it were a SASL mechanism. This facility is called GS2 -- a moniker that indicates that this is the second GSS-API->SASL mechanism bridge. The original GSS-API->SASL mechanism bridge was specified by [RFC2222], now [RFC4752]; we shall sometimes refer to the original bridge as GS1 in this document.

All GSS-API mechanisms are implicitly registered for use within SASL by this specification. The SASL mechanisms defined in this document are known as the GS2 family of mechanisms.

The GS1 bridge failed to gain wide deployment for any GSS-API mechanism other than "The Kerberos Version 5 GSS-API Mechanism" [RFC1964] [RFC4121], and has a number of problems that led us to desire a new bridge. Specifically, a) GS1 was not round-trip optimized and b) GS1 did not support channel binding [RFC5056]. These problems and the opportunity to create the next SASL password-based mechanism, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms" [RFC5802], as a GSS-API mechanism used by SASL applications via GS2, provide the motivation for GS2.

In particular, the current consensus of the SASL community appears to be that SASL "security layers" (i.e., confidentiality and integrity protection of application data after authentication) are too complex and redundant because SASL applications tend to have an option to run over a Transport Layer Security (TLS) [RFC5246] channel. Use of SASL security layers is best replaced with channel binding to a TLS channel.

GS2 is designed to be as simple as possible. It adds to GSS-API security context token exchanges only the bare minimum to support SASL semantics and negotiation of use of channel binding. Specifically, GS2 adds a small header (a few bytes plus the length of the client-requested SASL authorization identity) to the initial GSS-API context token and to the application channel binding data. GS2 uses SASL mechanism negotiation to implement channel binding negotiation. Security-relevant GS2 plaintext is protected via the use of GSS-API channel binding. Additionally, to simplify the

implementation of GS2 mechanisms for implementors who will not implement a GSS-API framework, we compress the initial security context token header required by [RFC2743], Section 3.1.

GS2 does not protect any plaintext exchanged outside GS2, such as SASL mechanism negotiation plaintext, or application messages following authentication. But using channel binding to a secure channel over which all SASL and application plaintext is sent will cause all that plaintext to be authenticated.

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].

The document uses many terms and function names defined in [RFC2743], as updated by [RFC5554].

3. Mechanism Name

There are two SASL mechanism names for any GSS-API mechanism used through this facility. One denotes that the server supports channel binding. The other denotes that it does not.

The SASL mechanism name for a GSS-API mechanism is that which is provided by that mechanism when it was specified, if one was specified. This name denotes that the server does not support channel binding. Add the suffix "-PLUS" and the resulting name denotes that the server does support channel binding. SASL implementations can use the GSS_Inquire_SASLname_for_mech call (see below) to query for the SASL mechanism name of a GSS-API mechanism.

If the GSS_Inquire_SASLname_for_mech interface is not used, the GS2 implementation needs some other mechanism to map mechanism Object Identifiers (OIDs) to SASL names internally. In this case, the implementation can only support the mechanisms for which it knows the SASL name. If GSS_Inquire_SASLname_for_mech() fails and the GS2 implementation cannot map the OID to a SASL mechanism name via some other means, then the GS2 implementation MUST NOT use the given GSS-API mechanism.

3.1. Generating SASL Mechanism Names from GSS-API OIDs

For GSS-API mechanisms whose SASL names are not defined together with the GSS-API mechanism or in this document, the SASL mechanism name is concatenation of the string "GS2-" and the Base32 encoding [RFC4648] (with an uppercase alphabet) of the first 55 bits of the binary SHA-1

hash [FIPS.180-1.1995] string computed over the ASN.1 DER encoding [CCITT.X690.2002], including the tag and length octets, of the GSS-API mechanism's Object Identifier. The Base32 rules on padding characters and characters outside of the Base32 alphabet are not relevant to this use of Base32. If any padding or non-alphabet characters are encountered, the name is not a GS2 family mechanism name. This name denotes that the server does not support channel binding. Add the suffix "-PLUS" and the resulting name denotes that the server does support channel binding.

A GS2 mechanism that has a non-OID-derived SASL mechanism name is said to have a "user-friendly SASL mechanism name".

3.2. Computing Mechanism Names Manually

The hash-derived GS2 SASL mechanism name may be computed manually. This is useful when the set of supported GSS-API mechanisms is known in advance. This eliminates the need to implement Base32, SHA-1, and DER in the SASL mechanism. The computed mechanism name can be used directly in the implementation, and the implementation need not be concerned if the mechanism is part of a mechanism family.

3.3. Examples

The OID for the Simple Public-Key GSS-API Mechanism (SPKM-1) [RFC2025] is 1.3.6.1.5.5.1.1. The ASN.1 DER encoding of the OID, including the tag and length, is (in hex) 06 07 2b 06 01 05 05 01 01. The SHA-1 hash of the ASN.1 DER encoding is (in hex) 1c f8 f4 2b 5a 9f 80 fa e9 f8 31 22 6d 5d 9d 56 27 86 61 ad. Convert the first 7 octets to binary, drop the last bit, and re-group them in groups of 5, and convert them back to decimal, which results in these computations:

hex:
1c f8 f4 2b 5a 9f 80

binary:
00011100 11111000 11110100 00101011 01011010
10011111 1000000

binary in groups of 5:
00011 10011 11100 01111 01000 01010 11010 11010
10011 11110 00000

decimal of each group:
3 19 28 15 8 10 26 26 19 30 0

base32 encoding:
D T 4 P I K 2 2 T 6 A

The last step translates each decimal value using table 3 in Base32 [RFC4648]. Thus, the SASL mechanism name for the SPKM-1 GSSAPI mechanism is "GS2-DT4PIK22T6A".

The OID for the Kerberos V5 GSS-API mechanism [RFC1964] is 1.2.840.113554.1.2.2 and its DER encoding is (in hex) 06 09 2A 86 48 86 F7 12 01 02 02. The SHA-1 hash is 82 d2 73 25 76 6b d6 c8 45 aa 93 25 51 6a fc ff 04 b0 43 60. Convert the 7 octets to binary, drop the last bit, and re-group them in groups of 5, and convert them back to decimal, which results in these computations:

hex:
82 d2 73 25 76 6b d6

binary:
10000010 11010010 01110011 00100101 01110110
01101011 1101011

binary in groups of 5:
10000 01011 01001 00111 00110 01001 01011 10110
01101 01111 01011

decimal of each group:
16 11 9 7 6 9 11 22 13 15 11

base32 encoding:
Q L J H G J L W N P L

The last step translates each decimal value using table 3 in Base32 [RFC4648]. Thus, the SASL mechanism name for the Kerberos V5 GSS-API mechanism would be "GS2-QLJHGJLWNPL" and (because this mechanism supports channel binding) "GS2-QLJHGJLWNPL-PLUS". Instead, the next section assigns the Kerberos V5 mechanism a non-hash-derived mechanism name.

3.4. Grandfathered Mechanism Names

Some older GSS-API mechanisms were not specified with a SASL GS2 mechanism name. Using a shorter name can be useful, nonetheless. We specify the names "GS2-KRB5" and "GS2-KRB5-PLUS" for the Kerberos V5 mechanism, to be used as if the original specification documented it, see Section 15.

4. SASL Authentication Exchange Message Format

During the SASL authentication exchange for GS2, a number of messages following the following format are sent between the client and server. On success, this number is the same as the number of context tokens that the GSS-API mechanism would normally require in order to establish a security context. On failures, the exchange can be terminated early by any party.

When using a GS2 mechanism the SASL client is always a GSS-API initiator and the SASL server is always a GSS-API acceptor. The client calls GSS_Init_sec_context and the server calls GSS_Accept_sec_context.

All the SASL authentication messages exchanged are exactly the same as the security context tokens of the GSS-API mechanism, except for the initial security context token.

The client and server MAY send GSS-API error tokens (tokens output by GSS_Init_sec_context() or GSS_Accept_sec_context() when the major status code is other than GSS_S_COMPLETE or GSS_S_CONTINUE_NEEDED). As this indicates an error condition, after sending the token, the sending side should fail the authentication.

The initial security context token is modified as follows:

- The initial context token header (see Section 3.1 of [RFC2743]) MUST be removed if present. If the header is not present, the client MUST send a "gs2-nonstd-flag" flag (see below). On the server side, this header MUST be recomputed and restored prior to passing the token to GSS_Accept_sec_context, except when the "gs2-nonstd-flag" is sent.

- A GS2 header MUST be prefixed to the resulting initial context token. This header has the form "gs2-header" given below in ABNF [RFC5234].

The figure below describes the permissible attributes, their use, and the format of their values. All attribute names are single US-ASCII letters and are case sensitive.

```
   UTF8-1-safe    = %x01-2B / %x2D-3C / %x3E-7F
                    ;; As UTF8-1 in RFC 3629 except
                    ;; NUL, "=", and ",".
   UTF8-2         = <as defined in RFC 3629 (STD 63)>
   UTF8-3         = <as defined in RFC 3629 (STD 63)>
   UTF8-4         = <as defined in RFC 3629 (STD 63)>
   UTF8-char-safe = UTF8-1-safe / UTF8-2 / UTF8-3 / UTF8-4

   saslname       = 1*(UTF8-char-safe / "=2C" / "=3D")
   gs2-authzid    = "a=" saslname
                     ;; GS2 has to transport an authzid since
                     ;; the GSS-API has no equivalent
   gs2-nonstd-flag = "F"
                     ;; "F" means the mechanism is not a
                     ;; standard GSS-API mechanism in that the
                     ;; RFC 2743, Section 3.1 header was missing
   cb-name         = 1*(ALPHA / DIGIT / "." / "-")
                     ;; See RFC 5056, Section 7.
   gs2-cb-flag     = ("p=" cb-name) / "n" / "y"
                     ;; GS2 channel binding (CB) flag
                     ;; "p" -> client supports and used CB
                     ;; "n" -> client does not support CB
                     ;; "y" -> client supports CB, thinks the server
                     ;;           does not
   gs2-header = [gs2-nonstd-flag ","] gs2-cb-flag "," [gs2-authzid] ","
                      ;; The GS2 header is gs2-header.
```

When the "gs2-nonstd-flag" flag is present, the client did not find/remove a token header ([RFC2743], Section 3.1) from the initial token returned by GSS_Init_sec_context. This signals to the server that it MUST NOT re-add the data that is normally removed by the client.

The "gs2-cb-flag" signals the channel binding mode. One of "p", "n", or "y" is used. A "p" means the client supports and used a channel binding, and the name of the channel binding type is indicated. An "n" means that the client does not support channel binding. A "y" means the client supports channel binding, but believes the server does not support it, so it did not use a channel binding. See the next section for more details.

The "gs2-authzid" holds the SASL authorization identity. It is encoded using UTF-8 [RFC3629] with three exceptions:

- The NUL character is forbidden as required by section 3.4.1 of [RFC4422].
- The server MUST replace any "," (comma) in the string with "=2C".

- The server MUST replace any "=" (equals) in the string with "=3D".

Upon receipt of this value, the server verifies its correctness according to the used SASL protocol profile. Failed verification results in a failed authentication exchange.

5. Channel Bindings

GS2 supports channel binding to external secure channels, such as TLS. Clients and servers may or may not support channel binding; therefore, the use of channel binding is negotiable. However, GS2 does not provide security layers; therefore, it is imperative that GS2 provide integrity protection for the negotiation of channel binding.

Use of channel binding is negotiated as follows:

- Servers that support the use of channel binding SHOULD advertise both the non-PLUS and PLUS-variant of each GS2 mechanism name. If the server cannot support channel binding, it SHOULD advertise only the non-PLUS-variant. If the server would never succeed in the authentication of the non-PLUS-variant due to policy reasons, it MUST advertise only the PLUS-variant.

- If the client supports channel binding and the server does not appear to (i.e., the client did not see the -PLUS name advertised by the server), then the client MUST NOT use an "n" gs2-cb-flag.

- Clients that support mechanism negotiation and channel binding MUST use a "p" gs2-cb-flag when the server offers the PLUS-variant of the desired GS2 mechanism.

- If the client does not support channel binding, then it MUST use an "n" gs2-cb-flag. Conversely, if the client requires the use of channel binding then it MUST use a "p" gs2-cb-flag. Clients that do not support mechanism negotiation never use a "y" gs2-cb-flag, they use either "p" or "n" according to whether they require and support the use of channel binding or whether they do not, respectively.

- The client generates the chan_bindings input parameter for GSS_Init_sec_context as described below.

- Upon receipt of the initial authentication message, the server checks the gs2-cb-flag in the GS2 header and constructs a chan_bindings parameter for GSS_Accept_sec_context as described below. If the client channel binding flag was "y" and the server did advertise support for channel bindings (by advertising the

PLUS-variant of the mechanism chosen by the client), then the server MUST fail authentication. If the client channel binding flag was "p" and the server does not support the indicated channel binding type, then the server MUST fail authentication.

- If the client used an "n" gs2-cb-flag and the server requires the use of channel binding, then the server MUST fail authentication.

FLAG	CLIENT CB SUPPORT	SERVER CB SUPPORT	DISPOSITION
n	no support	N/A	If server disallows non-channel-bound authentication, then fail
y	Yes, not required	No	Authentication may succeed; CB not used
y	Yes, not required	Yes	Authentication must fail
p	Yes	Yes	Authentication may succeed, with CB used
p	Yes	No	Authentication will fail
N/A	Yes, required	No	Client does not even try

For more discussion of channel bindings, and the syntax of the channel binding data for various security protocols, see [RFC5056].

5.1. Content of GSS-CHANNEL-BINDINGS Structure

The calls to GSS_Init_sec_context and GSS_Accept_sec_context take a chan_bindings parameter. The value is a GSS-CHANNEL-BINDINGS structure [RFC5554].

The initiator-address-type and acceptor-address-type fields of the GSS-CHANNEL-BINDINGS structure MUST be set to 0. The initiator-address and acceptor-address fields MUST be the empty string.

The application-data field MUST be set to the gs2-header, excluding the initial [gs2-nonstd-flag ","] part, concatenated with, when a gs2-cb-flag of "p" is used, the application’s channel binding data.

5.2. Default Channel Binding

A default channel binding type agreement process for all SASL application protocols that do not provide their own channel binding type agreement is provided as follows.

'tls-unique' is the default channel binding type for any application that doesn't specify one.

Servers MUST implement the "tls-unique" [RFC5929] channel binding type, if they implement any channel binding. Clients SHOULD implement the "tls-unique" channel binding type, if they implement any channel binding. Clients and servers SHOULD choose the highest-layer/innermost end-to-end TLS channel as the channel to which to bind.

Servers MUST choose the channel binding type indicated by the client, or fail authentication if they don't support it.

6. Examples

Example #1: a one round-trip GSS-API context token exchange, no channel binding, optional authzid given.

```
C: Request authentication exchange
S: Empty Challenge
C: n,a=someuser,<initial context token with standard
                 header removed>
S: Send reply context token as is
C: Empty message
S: Outcome of authentication exchange
```

Example #2: a one and one half round-trip GSS-API context token exchange, no channel binding.

```
C: Request authentication exchange
S: Empty Challenge
C: n,,<initial context token with standard
                 header removed>
S: Send reply context token as is
C: Send reply context token as is
S: Outcome of authentication exchange
```

Example #3: a two round-trip GSS-API context token exchange, no channel binding, no standard token header.

```
      C: Request authentication exchange
      S: Empty Challenge
      C: F,n,,<initial context token without
                          standard header>
      S: Send reply context token as is
      C: Send reply context token as is
      S: Send reply context token as is
      C: Empty message
      S: Outcome of authentication exchange
```

Example #4: using channel binding, optional authzid given.

```
      C: Request authentication exchange
      S: Empty Challenge
      C: p=tls-unique,a=someuser,<initial context token with standard
                             header removed>
      S: Send reply context token as is
      ...
```

Example #5: using channel binding.

```
      C: Request authentication exchange
      S: Empty Challenge
      C: p=tls-unique,,<initial context token with standard
                             header removed>
      S: Send reply context token as is
      ...
```

Example #6: using non-standard channel binding (requires out-of-band negotiation).

```
      C: Request authentication exchange
      S: Empty Challenge
      C: p=tls-server-end-point,,<initial context token with standard
                             header removed>
      S: Send reply context token as is
      ...
```

Example #7: client supports channel bindings but server does not, optional authzid given.

```
      C: Request authentication exchange
      S: Empty Challenge
      C: y,a=someuser,<initial
                       context token with standard header removed>
      S: Send reply context token as is
      ...
```

GSS-API authentication is always initiated by the client. The SASL framework allows either the client or the server to initiate authentication. In GS2, the server will send an initial empty challenge (zero-byte string) if it has not yet received a token from the client. See Section 3 of [RFC4422].

7. Authentication Conditions

Authentication MUST NOT succeed if any one of the following conditions are true:

- If GSS_Init/Accept_sec_context returns anything other than GSS_S_CONTINUE_NEEDED or GSS_S_COMPLETE.
- If the client's initial GS2 header does not match the ABNF.
- In particular, if the initial character of the client message is anything except "F", "p", "n", or "y".
- If the client's GS2 channel binding flag was "y" and the server supports channel bindings.
- If the client's GS2 channel binding flag was "p" and the server does not support the indicated channel binding.
- If the client requires use of channel binding and the server did not advertise support for channel binding.
- If authorization of client principal (i.e., src_name in GSS_Accept_sec_context) to requested authzid failed.
- If the client is not authorized to the requested authzid or an authzid could not be derived from the client's initiator principal name.

8. GSS-API Parameters

GS2 does not use any GSS-API per-message tokens. Therefore, the per-message token ret_flags from GSS_Init_sec_context() and GSS_Accept_sec_context() are irrelevant; implementations SHOULD NOT set the per-message req_flags.

The mutual_req_flag MUST be set. Clients MUST check that the corresponding ret_flag is set when the context is fully established, else authentication MUST fail.

Use or non-use of deleg_req_flag and anon_req_flag is an implementation-specific detail. SASL and GS2 implementors are encouraged to provide programming interfaces by which clients may choose to delegate credentials and by which servers may receive them. SASL and GS2 implementors are encouraged to provide programming interfaces that provide a good mapping of GSS-API naming options.

9. Naming

There is no requirement that any particular GSS-API name-types be used. However, typically, SASL servers will have host-based acceptor principal names (see [RFC2743], Section 4.1) and clients will typically have username initiator principal names (see [RFC2743], Section 4.2). When a host-based acceptor principal name is used ("service@hostname"), "service" is the service name specified in the protocol's profile and "hostname" is the fully qualified host name of the server.

10. GSS_Inquire_SASLname_for_mech Call

We specify a new GSS-API utility function to allow SASL implementations to more efficiently identify the GSS-API mechanism to which a particular SASL mechanism name refers.

Inputs:

- desired_mech OBJECT IDENTIFIER

Outputs:

- major_status INTEGER
- minor_status INTEGER
- sasl_mech_name UTF-8 STRING -- SASL name for this mechanism; caller must release with GSS_Release_buffer()

- mech_name UTF-8 STRING -- name of this mechanism, possibly localized; caller must release with GSS_Release_buffer()

- mech_description UTF-8 STRING -- possibly localized description of this mechanism; caller must release with GSS_Release_buffer()

Return major_status codes:

- GSS_S_COMPLETE indicates successful completion, and that output parameters holds correct information.

- GSS_S_BAD_MECH indicates that a desired_mech was unsupported by the GSS-API implementation.

- GSS_S_FAILURE indicates that the operation failed for reasons unspecified at the GSS-API level.

The GSS_Inquire_SASLname_for_mech call is used to get the SASL mechanism name for a GSS-API mechanism. It also returns a name and description of the mechanism in user-friendly form.

The output variable sasl_mech_name will hold the IANA registered mechanism name for the GSS-API mechanism, or if none is registered, a mechanism name computed from the OID as described in Section 3.1 of this document.

10.1. gss_inquire_saslname_for_mech

The C binding for the GSS_Inquire_SASLname_for_mech call is as follows.

As mentioned in [RFC2744], routines may return GSS_S_FAILURE, indicating an implementation-specific or mechanism-specific error condition, further details of which are reported via the minor_status parameter.

```
OM_uint32 gss_inquire_saslname_for_mech(
  OM_uint32     *minor_status,
  const gss_OID  desired_mech,
  gss_buffer_t   sasl_mech_name,
  gss_buffer_t   mech_name,
  gss_buffer_t   mech_description
);
```

Purpose:

Output the SASL mechanism name of a GSS-API mechanism.
It also returns a name and description of the mechanism in a
user-friendly form.

Parameters:

minor_status	Integer, modify Mechanism-specific status code.
desired_mech	OID, read Identifies the GSS-API mechanism to query.
sasl_mech_name	buffer, character-string, modify, optional Buffer to receive SASL mechanism name. The application must free storage associated with this name after use with a call to gss_release_buffer().
mech_name	buffer, character-string, modify, optional Buffer to receive human-readable mechanism name. The application must free storage associated with this name after use with a call to gss_release_buffer().
mech_description	buffer, character-string, modify, optional Buffer to receive description of mechanism. The application must free storage associated with this name after use with a call to gss_release_buffer().
Function value:	GSS status code:
GSS_S_COMPLETE	Successful completion.
GSS_S_BAD_MECH	The desired_mech OID is unsupported.

11. GSS_Inquire_mech_for_SASLname Call

To allow SASL clients to more efficiently identify to which GSS-API mechanism a particular SASL mechanism name refers, we specify a new GSS-API utility function for this purpose.

Inputs:

- sasl_mech_name UTF-8 STRING -- SASL name of mechanism.

Outputs:

- major_status INTEGER
- minor_status INTEGER
- mech_type OBJECT IDENTIFIER -- must be explicit mechanism, and not "default" specifier. Caller should treat as read-only and should not attempt to release.

Return major_status codes:

- GSS_S_COMPLETE indicates successful completion, and that output parameters holds correct information.
- GSS_S_BAD_MECH indicates that no supported GSS-API mechanism had the indicated sasl_mech_name.
- GSS_S_FAILURE indicates that the operation failed for reasons unspecified at the GSS-API level.

The GSS_Inquire_mech_for_SASLname call is used to get the GSS-API mechanism OID associated with a SASL mechanism name.

11.1. gss_inquire_mech_for_saslname

The C binding for the GSS_Inquire_mech_for_SASLname call is as follows.

As mentioned in [RFC2744], routines may return GSS_S_FAILURE, indicating an implementation-specific or mechanism-specific error condition, further details of which are reported via the minor_status parameter.

```
OM_uint32 gss_inquire_mech_for_saslname(
  OM_uint32           *minor_status,
  const gss_buffer_t   sasl_mech_name,
  gss_OID             *mech_type
);
```

Purpose:

Output GSS-API mechanism OID of mechanism associated with given sasl_mech_name.

Parameters:

minor_status	Integer, modify Mechanism-specific status code.
sasl_mech_name	buffer, character-string, read Buffer with SASL mechanism name.
mech_type	OID, modify, optional Actual mechanism used. The OID returned via this parameter will be a pointer to static storage that should be treated as read-only. In particular, the application should not attempt to free it. Specify NULL if not required.
Function value:	GSS status code:
GSS_S_COMPLETE	Successful completion.
GSS_S_BAD_MECH	There is no GSS-API mechanism known as sasl_mech_name.

12. Security Layers

GS2 does not support SASL security layers. Applications that need integrity or confidentiality protection can use either channel binding to a secure external channel or another SASL mechanism that does provide security layers.

13. Interoperability with the SASL GSSAPI Mechanism

The Kerberos V5 GSS-API [RFC1964] mechanism is currently used in SASL under the name GSSAPI, see [RFC4752]. The Kerberos V5 mechanism may also be used with the GS2 family. This causes an interoperability problem, which is discussed and resolved below.

13.1. The Interoperability Problem

The SASL "GSSAPI" mechanism is not wire compatible with the Kerberos V GSS-API mechanism used as a SASL GS2 mechanism.

If a client (or server) only support Kerberos V5 under the "GSSAPI" name, and the server (or client) only support Kerberos V5 under the GS2 family, the mechanism negotiation will fail.

13.2. Resolving the Problem

If the Kerberos V5 mechanism is supported under GS2 in a server, the server SHOULD also support Kerberos V5 through the "GSSAPI" mechanism, to avoid interoperability problems with older clients.

Reasons for violating this recommendation may include security considerations regarding the absent features in the GS2 mechanism. The SASL "GSSAPI" mechanism lacks support for channel bindings, which means that using an external secure channel may not be sufficient protection against active attackers (see [RFC5056] and [MITM]).

13.3. Additional Recommendations

If the application requires SASL security layers, then it MUST use the SASL "GSSAPI" mechanism [RFC4752] instead of "GS2-KRB5" or "GS2-KRB5-PLUS".

If the application can use channel binding to an external channel, then it is RECOMMENDED that it select Kerberos V5 through the GS2 mechanism rather than the "GSSAPI" mechanism.

14. GSS-API Mechanisms That Negotiate Other Mechanisms

A GSS-API mechanism that negotiates other mechanisms will interact badly with the SASL mechanism negotiation. There are two problems. The first is an interoperability problem and the second is a security concern. The problems are described and resolved below.

14.1. The Interoperability Problem

If a client implements GSS-API mechanism X, potentially negotiated through a GSS-API mechanism Y, and the server also implements GSS-API mechanism X negotiated through a GSS-API mechanism Z, the authentication negotiation will fail.

14.2. Security Problem

If a client's policy is to first prefer GSSAPI mechanism X, then non-GSSAPI mechanism Y, then GSSAPI mechanism Z, and if a server supports mechanisms Y and Z but not X, then if the client attempts to negotiate mechanism X by using a GSS-API mechanism that negotiates other mechanisms (such as Simple and Protected GSS-API Negotiation (SPNEGO) [RFC4178]), it may end up using mechanism Z when it ideally should have used mechanism Y. For this reason, the use of GSS-API mechanisms that negotiate other mechanisms is disallowed under GS2.

14.3. Resolving the Problems

GSS-API mechanisms that negotiate other mechanisms MUST NOT be used with the GS2 SASL mechanism. Specifically, SPNEGO [RFC4178] MUST NOT be used as a GS2 mechanism. To make this easier for SASL implementations, we assign a symbolic SASL mechanism name to the SPNEGO GSS-API mechanism, "SPNEGO". SASL client implementations MUST NOT choose the SPNEGO mechanism under any circumstances.

The GSS_C_MA_MECH_NEGO attribute of GSS_Inquire_attrs_for_mech [RFC5587] can be used to identify such mechanisms.

15. IANA Considerations

The IANA has registered a SASL mechanism family as per [RFC4422] using the following information.

```
Subject: Registration of SASL mechanism family GS2-*
SASL mechanism prefix: GS2-
Security considerations: RFC 5801
Published specification: RFC 5801
Person & email address to contact for further information:
  Simon Josefsson <simon@josefsson.org>
Intended usage: COMMON
Owner/Change controller: iesg@ietf.org
Note: Compare with the GSSAPI and GSS-SPNEGO mechanisms.
```

The IANA is advised that SASL mechanism names starting with "GS2-" are reserved for SASL mechanisms that conform to this document. The IANA has placed a statement to that effect in the SASL Mechanisms registry.

The IANA is further advised that GS2 SASL mechanism names MUST NOT end in "-PLUS" except as a version of another mechanism name simply suffixed with "-PLUS".

The SASL names for the Kerberos V5 GSS-API mechanism [RFC4121] [RFC1964] used via GS2 SHALL be "GS2-KRB5" and "GS2-KRB5-PLUS".

The SASL names for the SPNEGO GSS-API mechanism used via GS2 SHALL be "SPNEGO" and "SPNEGO-PLUS". As described in Section 14, the SASL "SPNEGO" and "SPNEGO-PLUS" MUST NOT be used. These names are provided as a convenience for SASL library implementors.

16. Security Considerations

Security issues are also discussed throughout this memo.

The security provided by a GS2 mechanism depends on the security of the GSS-API mechanism. The GS2 mechanism family depends on channel binding support, so GSS-API mechanisms that do not support channel binding cannot be successfully used as SASL mechanisms via the GS2 bridge.

Because GS2 does not support security layers, it is strongly RECOMMENDED that channel binding to a secure external channel be used. Successful channel binding eliminates the possibility of man-in-the-middle (MITM) attacks, provided that the external channel and its channel binding data are secure and that the GSS-API mechanism used is secure. Authentication failure because of channel binding

failure may indicate that an MITM attack was attempted, but note that a real MITM attacker would likely attempt to close the connection to the client or simulate network partition; thus, MITM attack detection is heuristic.

Use of channel binding will also protect the SASL mechanism negotiation -- if there is no MITM, then the external secure channel will have protected the SASL mechanism negotiation.

The channel binding data MAY be sent (by the actual GSS-API mechanism used) without confidentiality protection and knowledge of it is assumed to provide no advantage to an MITM (who can, in any case, compute the channel binding data independently). If the external channel does not provide confidentiality protection and the GSS-API mechanism does not provide confidentiality protection for the channel binding data, then passive attackers (eavesdroppers) can recover the channel binding data, see [RFC5056].

When constructing the input_name_string for GSS_Import_name with the GSS_C_NT_HOSTBASED_SERVICE name type, the client SHOULD NOT canonicalize the server’s fully qualified domain name using an insecure or untrusted directory service, such as the Domain Name System [RFC1034] without DNS Security (DNSSEC) [RFC4033].

SHA-1 is used to derive SASL mechanism names, but no traditional cryptographic properties are required -- the required property is that the truncated output for distinct inputs are different for practical input values. GS2 does not use any other cryptographic algorithm. Therefore, GS2 is "algorithm agile", or, as agile as the GSS-API mechanisms that are available for use in SASL applications via GS2.

GS2 does not protect against downgrade attacks of channel binding types. Negotiation of channel binding type was intentionally left out of scope for this document.

The security considerations of SASL [RFC4422], the GSS-API [RFC2743], channel binding [RFC5056], any external channels (such as TLS, [RFC5246], channel binding types (see the IANA channel binding type registry), and GSS-API mechanisms (such as the Kerberos V5 mechanism [RFC4121] [RFC1964]), also apply.

17. Acknowledgements

The history of GS2 can be traced to the "GSSAPI" mechanism originally specified by RFC 2222. This document was derived from [SASL-GSSAPI], which was prepared by Alexey Melnikov with significant contributions from John G. Myers, although the majority of this document has been rewritten by the current authors.

Contributions of many members of the SASL mailing list are gratefully acknowledged. In particular, ideas and feedback from Pasi Eronen, Sam Hartman, Jeffrey Hutzelman, Alexey Melnikov, and Tom Yu improved the document and the protocol. Other suggestions to the documents were made by Spencer Dawkins, Ralph Droms, Adrian Farrel, Robert Sparks, and Glen Zorn.

18. References

18.1. Normative References

[FIPS.180-1.1995] National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-1, April 1995, <http://www.itl.nist.gov/fipspubs/fip180-1.htm>.

[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.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5554] Williams, N., "Clarifications and Extensions to the Generic Security Service Application Program Interface (GSS-API) for the Use of Channel Bindings", RFC 5554, May 2009.

[CCITT.X690.2002] International Telephone and Telegraph Consultative Committee, "ASN.1 encoding rules: Specification of basic encoding Rules (BER), Canonical encoding rules (CER) and Distinguished encoding rules (DER)", CCITT Recommendation X.690, July 2002.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, July 2010.

18.2. Informative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996.

[RFC2025] Adams, C., "The Simple Public-Key GSS-API Mechanism (SPKM)", RFC 2025, October 1996.

[RFC2222] Myers, J., "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997.

[RFC2744] Wray, J., "Generic Security Service API Version 2 : C-bindings", RFC 2744, January 2000.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 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.

[RFC4752] Melnikov, A., "The Kerberos V5 ("GSSAPI") Simple Authentication and Security Layer (SASL) Mechanism", RFC 4752, November 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5587] Williams, N., "Extended Generic Security Service Mechanism Inquiry APIs", RFC 5587, July 2009.

[RFC5802] Menon-Sen, A., Melnikov, A., Newman, C., and N. Williams, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms", RFC 5802, July 2010.

[MITM] Asokan, N., Niemi, V., and K. Nyberg, "Man-in-the-Middle in Tunnelled Authentication", in 11th Security Protocols Workshop, 2002.

[SASL-GSSAPI] Melnikov, A., "The Kerberos V5 ("GSSAPI") SASL mechanism", Work in Progress, March 2005.

Authors' Addresses

Simon Josefsson
SJD AB
Hagagatan 24
Stockholm 113 47
SE

EMail: simon@josefsson.org
URI: http://josefsson.org/

Nicolas Williams
Oracle
5300 Riata Trace Ct
Austin, TX 78727
USA

EMail: Nicolas.Williams@oracle.com