NAME

gss_init_sec_context - Initiate a security context with a peer application

SYNOPSIS

#include <gssapi/gssapi.h>

OM_uint32
gss_init_sec_context(*OM_uint32 *minor_status, const gss_cred_id_t initiator_cred_handle, gss_ctx_id_t *context_handle, const gss_name_t target_name, const gss_OID mech_type, OM_uint32 req_flags, OM_uint32 time_req, const gss_channel_bindings_t input_chan_bindings, const gss_buffer_t input_token, gss_OID *actual_mech_type, gss_buffer_t output_token, OM_uint32 *ret_flags, OM_uint32 *time_rec*);

DESCRIPTION

Initiates the establishment of a security context between the application and a remote peer. Initially, the input_token parameter should be specified either as GSS_C_NO_BUFFER, or as a pointer to a gss_buffer_desc object whose length field contains the value zero. The routine may return a output_token which should be transferred to the peer application, where the peer application will present it to gss_accept_sec_context(3). If no token need be sent, **gss_init_sec_context**() will indicate this by setting the length field of the output_token argument to zero. To complete the context establishment, one or more reply tokens may be required from the peer application; if so, **gss_init_sec_context**() will return a status containing the supplementary information bit GSS_S_CONTINUE_NEEDED. In this case, **gss_init_sec_context**() should be called again when the reply token is received from the peer application, passing the reply token to **gss_init_sec_context**() via the input_token parameters.

Portable applications should be constructed to use the token length and return status to determine whether a token needs to be sent or waited for. Thus a typical portable caller should always invoke **gss_init_sec_context**() within a loop:

```
int context_established = 0;
gss_ctx_id_t context_hdl = GSS_C_NO_CONTEXT;
      ...
input_token->length = 0;

while (!context_established) {
  maj_stat = gss_init_sec_context(&min_stat,
                                  cred_hdl,
                                  &context_hdl,
                                  target_name,
                                  desired_mech,
```

```
                                        desired_services,
                                        desired_time,
                                        input_bindings,
                                        input_token,
                                        &actual_mech,
                                        output_token,
                                        &actual_services,
                                        &actual_time);
  if (GSS_ERROR(maj_stat)) {
    report_error(maj_stat, min_stat);
  };

  if (output_token->length != 0) {
    send_token_to_peer(output_token);
    gss_release_buffer(&min_stat, output_token)
  };
  if (GSS_ERROR(maj_stat)) {

    if (context_hdl != GSS_C_NO_CONTEXT)
      gss_delete_sec_context(&min_stat,
                             &context_hdl,
                             GSS_C_NO_BUFFER);
    break;
  };

  if (maj_stat & GSS_S_CONTINUE_NEEDED) {
    receive_token_from_peer(input_token);
  } else {
    context_established = 1;
  };
};
```

Whenever the routine returns a major status that includes the value GSS_S_CONTINUE_NEEDED, the context is not fully established and the following restrictions apply to the output parameters:

- The value returned via the *time_rec* parameter is undefined Unless the accompanying *ret_flags* parameter contains the bit GSS_C_PROT_READY_FLAG, indicating that per-message services may be applied in advance of a successful completion status, the value returned via the *actual_mech_type* parameter is undefined until the routine returns a major status value of GSS_S_COMPLETE.

- The values of the GSS_C_DELEG_FLAG, GSS_C_MUTUAL_FLAG, GSS_C_REPLAY_FLAG, GSS_C_SEQUENCE_FLAG, GSS_C_CONF_FLAG, GSS_C_INTEG_FLAG and GSS_C_ANON_FLAG bits returned via the *ret_flags* parameter should contain the values that the implementation expects would be valid if context establishment were to succeed. In particular, if the application has requested a service such as delegation or anonymous authentication via the *req_flags* argument, and such a service is unavailable from the underlying mechanism, **gss_init_sec_context**() should generate a token that will not provide the service, and indicate via the *ret_flags* argument that the service will not be supported. The application may choose to abort the context establishment by calling gss_delete_sec_context(3) (if it cannot continue in the absence of the service), or it may choose to transmit the token and continue context establishment (if the service was merely desired but not mandatory).

- The values of the GSS_C_PROT_READY_FLAG and GSS_C_TRANS_FLAG bits within *ret_flags* should indicate the actual state at the time **gss_init_sec_context**() returns, whether or not the context is fully established.

- GSS-API implementations that support per-message protection are encouraged to set the GSS_C_PROT_READY_FLAG in the final *ret_flags* returned to a caller (i.e. when accompanied by a GSS_S_COMPLETE status code). However, applications should not rely on this behavior as the flag was not defined in Version 1 of the GSS-API. Instead, applications should determine what per-message services are available after a successful context establishment according to the GSS_C_INTEG_FLAG and GSS_C_CONF_FLAG values.

- All other bits within the *ret_flags* argument should be set to zero.

If the initial call of **gss_init_sec_context**() fails, the implementation should not create a context object, and should leave the value of the *context_handle* parameter set to GSS_C_NO_CONTEXT to indicate this. In the event of a failure on a subsequent call, the implementation is permitted to delete the "half-built" security context (in which case it should set the *context_handle* parameter to GSS_C_NO_CONTEXT), but the preferred behavior is to leave the security context untouched for the application to delete (using gss_delete_sec_context(3)).

During context establishment, the informational status bits GSS_S_OLD_TOKEN and GSS_S_DUPLICATE_TOKEN indicate fatal errors, and GSS-API mechanisms should always return them in association with a routine error of GSS_S_FAILURE. This requirement for pairing did not exist in version 1 of the GSS-API specification, so applications that wish to run over version 1 implementations must special-case these codes.

PARAMETERS

minor_status Mechanism specific status code.

initiator_cred_handle — handle for credentials claimed. Supply GSS_C_NO_CREDENTIAL to act as a default initiator principal. If no default initiator is defined, the function will return GSS_S_NO_CRED.

context_handle — context handle for new context. Supply GSS_C_NO_CONTEXT for first call; use value returned by first call in continuation calls. Resources associated with this context-handle must be released by the application after use with a call to **gss_delete_sec_context**().

target_name — Name of target

mech_type — Object ID of desired mechanism. Supply GSS_C_NO_OID to obtain an implementation specific default

req_flags — Contains various independent flags, each of which requests that the context support a specific service option. Symbolic names are provided for each flag, and the symbolic names corresponding to the required flags should be logically-ORed together to form the bit-mask value. The flags are:

GSS_C_DELEG_FLAG

True Delegate credentials to remote peer

False Don’t delegate

GSS_C_MUTUAL_FLAG

True Request that remote peer authenticate itself

False Authenticate self to remote peer only

GSS_C_REPLAY_FLAG

True Enable replay detection for messages protected with gss_wrap(3) or gss_get_mic(3)

False Don’t attempt to detect replayed messages

GSS_C_SEQUENCE_FLAG

True Enable detection of out-of-sequence protected messages

False Don’t attempt to detect out-of-sequence messages

GSS_C_CONF_FLAG

True Request that confidentiality service be made available (via gss_wrap(3))

False No per-message confidentiality service is required.

GSS_C_INTEG_FLAG

True Request that integrity service be made available (via gss_wrap(3) or gss_get_mic(3))

False No per-message integrity service is required.

GSS_C_ANON_FLAG

True Do not reveal the initiator’s identity to the acceptor.

False Authenticate normally.

time_req Desired number of seconds for which context should remain valid. Supply 0 to request a default validity period.

input_chan_bindings Application-specified bindings. Allows application to securely bind channel identification information to the security context. Specify GSS_C_NO_CHANNEL_BINDINGS if channel bindings are not used.

input_token Token received from peer application. Supply GSS_C_NO_BUFFER, or a pointer to a buffer containing the value GSS_C_EMPTY_BUFFER on initial call.

actual_mech_type 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.

output_token token to be sent to peer application. If the length field of the returned buffer is zero, no token need be sent to the peer application. Storage associated with this

buffer must be freed by the application after use with a call to gss_release_buffer(3).

ret_flags — Contains various independent flags, each of which indicates that the context supports a specific service option. Specify NULL if not required. Symbolic names are provided for each flag, and the symbolic names corresponding to the required flags should be logically-ANDed with the *ret_flags* value to test whether a given option is supported by the context. The flags are:

GSS_C_DELEG_FLAG

- True — Credentials were delegated to the remote peer
- False — No credentials were delegated

GSS_C_MUTUAL_FLAG

- True — The remote peer has authenticated itself.
- False — Remote peer has not authenticated itself.

GSS_C_REPLAY_FLAG

- True — Replay of protected messages will be detected
- False — Replayed messages will not be detected

GSS_C_SEQUENCE_FLAG

- True — Out-of-sequence protected messages will be detected
- False — Out-of-sequence messages will not be detected

GSS_C_CONF_FLAG

- True — Confidentiality service may be invoked by calling gss_wrap(3) routine
- False — No confidentiality service (via gss_wrap(3)) available. gss_wrap(3) will provide message encapsulation, data-origin authentication and

integrity services only.

GSS_C_INTEG_FLAG

True Integrity service may be invoked by calling either gss_get_mic(3) or gss_wrap(3) routines.

False Per-message integrity service unavailable.

GSS_C_ANON_FLAG

True The initiator's identity has not been revealed, and will not be revealed if any emitted token is passed to the acceptor.

False The initiator's identity has been or will be authenticated normally.

GSS_C_PROT_READY_FLAG

True Protection services (as specified by the states of the GSS_C_CONF_FLAG and GSS_C_INTEG_FLAG) are available for use if the accompanying major status return value is either GSS_S_COMPLETE or GSS_S_CONTINUE_NEEDED.

False Protection services (as specified by the states of the GSS_C_CONF_FLAG and GSS_C_INTEG_FLAG) are available only if the accompanying major status return value is GSS_S_COMPLETE.

GSS_C_TRANS_FLAG

True The resultant security context may be transferred to other processes via a call to **gss_export_sec_context**().

False The security context is not transferable.

All other bits should be set to zero.

time_rec Number of seconds for which the context will remain valid. If the implementation does not support context expiration, the value GSS_C_INDEFINITE will be returned. Specify NULL if not required.

RETURN VALUES

GSS_S_COMPLETE	Successful completion
GSS_S_CONTINUE_NEEDED	Indicates that a token from the peer application is required to complete the context, and that gss_init_sec_context must be called again with that token.
GSS_S_DEFECTIVE_TOKEN	Indicates that consistency checks performed on the input_token failed
GSS_S_DEFECTIVE_CREDENTIAL	Indicates that consistency checks performed on the credential failed.
GSS_S_NO_CRED	The supplied credentials were not valid for context initiation, or the credential handle did not reference any credentials.
GSS_S_CREDENTIALS_EXPIRED	The referenced credentials have expired
GSS_S_BAD_BINDINGS	The input_token contains different channel bindings to those specified via the input_chan_bindings parameter
GSS_S_BAD_SIG	The input_token contains an invalid MIC, or a MIC that could not be verified
GSS_S_OLD_TOKEN	The input_token was too old. This is a fatal error during context establishment
GSS_S_DUPLICATE_TOKEN	The input_token is valid, but is a duplicate of a token already processed. This is a fatal error during context establishment.
GSS_S_NO_CONTEXT	Indicates that the supplied context handle did not refer to a valid context
GSS_S_BAD_NAMETYPE	The provided target_name parameter contained an invalid or unsupported type of name
GSS_S_BAD_NAME	The provided target_name parameter was ill-formed.
GSS_S_BAD_MECH	The specified mechanism is not supported by the provided

credential, or is unrecognized by the implementation.

SEE ALSO

gss_accept_sec_context(3), gss_delete_sec_context(3), gss_get_mic(3), gss_release_buffer(3), gss_wrap(3)

STANDARDS

RFC 2743 Generic Security Service Application Program Interface Version 2, Update 1

RFC 2744 Generic Security Service API Version 2 : C-bindings

HISTORY

The **gss_init_sec_context** function first appeared in FreeBSD 7.0.

AUTHORS

John Wray, Iris Associates

COPYRIGHT