NAME

gss_accept_sec_context - Accept a security context initiated by a peer application

SYNOPSIS

#include <gssapi/gssapi.h>

OM_uint32
gss_accept_sec_context(*OM_uint32 *minor_status*, *gss_ctx_id_t *context_handle*,
const gss_cred_id_t acceptor_cred_handle, *const gss_buffer_t input_token_buffer*,
const gss_channel_bindings_t input_chan_bindings, *const gss_name_t *src_name*,
*gss_OID *mech_type*, *gss_buffer_t output_token*, *OM_uint32 *ret_flags*, *OM_uint32 *time_rec*,
*gss_cred_id_t *delegated_cred_handle*);

DESCRIPTION

Allows a remotely initiated security context between the application and a remote peer to be established. The routine may return a *output_token* which should be transferred to the peer application, where the peer application will present it to gss_init_sec_context(3). If no token need be sent, **gss_accept_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_accept_sec_context**() will return a status flag of GSS_S_CONTINUE_NEEDED, in which case it should be called again when the reply token is received from the peer application, passing the token to **gss_accept_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_accept_sec_context**() within a loop:

```
gss_ctx_id_t context_hdl = GSS_C_NO_CONTEXT;

do {
  receive_token_from_peer(input_token);
  maj_stat = gss_accept_sec_context(&min_stat,
                                    &context_hdl,
                                    cred_hdl,
                                    input_token,
                                    input_bindings,
                                    &client_name,
                                    &mech_type,
                                    output_token,
                                    &ret_flags,
                                    &time_rec,
                                    &deleg_cred);
  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;
  };
} while (maj_stat & GSS_S_CONTINUE_NEEDED);
```

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 *mech_type* parameter may be 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.

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_accept_sec_context**() returns, whether or not the context is fully established.

Although this requires that GSS-API implementations 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), applications should not rely on this behavior as the flag was not defined in Version 1 of the GSS-API. Instead, applications should be prepared to use per-message services 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. While the routine returns GSS_S_CONTINUE_NEEDED, the values returned via the *ret_flags* argument indicate the services that the implementation expects to be available from the established context.

If the initial call of **gss_accept_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 (and the context_handle parameter) 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

context_handle
Context handle for new context. Supply GSS_C_NO_CONTEXT for first call; use value returned in subsequent calls. Once **gss_accept_sec_context**() has returned a value via this parameter, resources have been assigned to the corresponding context, and must be freed by the application after use with a call to gss_delete_sec_context(3).

acceptor_cred_handle
Credential handle claimed by context acceptor. Specify GSS_C_NO_CREDENTIAL to accept the context as a default principal. If GSS_C_NO_CREDENTIAL is specified, but no default acceptor principal is defined, GSS_S_NO_CRED will be returned.

input_token_buffer
Token obtained from remote application.

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

src_name
Authenticated name of context initiator. After use, this name should be deallocated by passing it to gss_release_name(3). If not required, specify NULL.

mech_type

Security mechanism used. The returned OID value will be a pointer into static storage, and should be treated as read-only by the caller (in particular, it does not need to be freed). If not required, specify NULL.

output_token

Token to be passed to peer application. If the length field of the returned token buffer is 0, then no token need be passed to the peer application. If a non-zero length field is returned, the associated storage must be freed after use by the application 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. If not needed, specify NULL. 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 Delegated credentials are available via the delegated_cred_handle parameter

False No credentials were delegated

GSS_C_MUTUAL_FLAG

True Remote peer asked for mutual authentication

False Remote peer did not ask for mutual authentication

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 the 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 does not wish to be authenticated; the *src_name* parameter (if requested) contains an anonymous internal name.

False The initiator has been 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 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(3).

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. Specify NULL if not required.

delegated_cred_handle
Credential handle for credentials received from context initiator. Only valid if GSS_C_DELEG_FLAG in *ret_flags* is true, in which case an explicit credential handle (i.e. not GSS_C_NO_CREDENTIAL) will be returned; if false, **gss_accept_context**() will set this parameter to GSS_C_NO_CREDENTIAL. If a credential handle is returned, the associated resources must be released by the application after use with a call to gss_release_cred(3). Specify NULL if not required.

minor_status
Mechanism specific status code.

RETURN VALUES

GSS_S_CONTINUE_NEEDED
Indicates that a token from the peer application is required to complete the context, and that gss_accept_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 acceptance, 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_NO_CONTEXT
Indicates that the supplied context handle did not refer to a valid context.

GSS_S_BAD_SIG The input_token contains an invalid MIC.

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_BAD_MECH The received token specified a mechanism that is not supported by the implementation or the provided credential.

SEE ALSO

gss_delete_sec_context(3), gss_export_sec_context(3), gss_get_mic(3), gss_init_sec_context(3), gss_release_buffer(3), gss_release_cred(3), gss_release_name(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_accept_sec_context** function first appeared in FreeBSD 7.0.

AUTHORS

John Wray, Iris Associates

COPYRIGHT

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.