

GSS-API Authentication Method for SOCKS Version 5

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Table of Contents

1. Purpose	1
2. Introduction	1
3. GSS-API Security Context Establishment	2
4. GSS-API Protection-level Options	5
5. GSS-API Per-message Protection	7
6. GSS-API Security Context Termination	8
7. References	8
8. Acknowledgments	8
9. Security Considerations	8
10. Author's Address	9

1. Purpose

The protocol specification for SOCKS Version 5 specifies a generalized framework for the use of arbitrary authentication protocols in the initial SOCKS connection setup. This document provides the specification for the SOCKS V5 GSS-API authentication protocol, and defines a GSS-API-based encapsulation for provision of integrity, authentication and optional confidentiality.

2. Introduction

GSS-API provides an abstract interface which provides security services for use in distributed applications, but isolates callers from specific security mechanisms and implementations.

GSS-API peers achieve interoperability by establishing a common security mechanism for security context establishment - either through administrative action, or through negotiation. GSS-API is specified in [RFC 1508], and [RFC 1509]. This specification is intended for use with implementations of GSS-API, and the emerging

GSS-API V2 specification.

The approach for use of GSS-API in SOCKS V5 is to authenticate the client and server by successfully establishing a GSS-API security context - such that the GSS-API encapsulates any negotiation protocol for mechanism selection, and the agreement of security service options.

The GSS-API enables the context initiator to know what security services the target supports for the chosen mechanism. The required level of protection is then agreed by negotiation.

The GSS-API per-message protection calls are subsequently used to encapsulate any further TCP and UDP traffic between client and server.

3. GSS-API Security Context Establishment

3.1 Preparation

Prior to use of GSS-API primitives, the client and server should be locally authenticated, and have established default GSS-API credentials.

The client should call `gss_import_name` to obtain an internal representation of the server name. For maximal portability the default `name_type` `GSS_C_NULL_OID` should be used to specify the default name space, and the input `name_string` should be treated by the client's code as an opaque name-space specific input.

For example, when using Kerberos V5 naming, the imported name may be of the form "SERVICE:socks@socks_server_hostname" where "socks_server_hostname" is the fully qualified host name of the server with all letters in lower case. Other mechanisms may, however, have different name forms, so the client should not make assumptions about the name syntax.

3.2 Client Context Establishment

The client should then call `gss_init_sec_context`, typically passing:

`GSS_C_NO_CREDENTIAL` into `cred_handle` to specify the default credential (for initiator usage),

`GSS_C_NULL_OID` into `mech_type` to specify the default mechanism,

GSS_C_NO_CONTEXT into context_handle to specify a NULL context (initially), and,

the previously imported server name into target_name.

The client must also specify its requirements for replay protection, delegation, and sequence protection via the gss_init_sec_context req_flags parameter. It is required by this specification that the client always requests these service options (i.e. passes GSS_C_MUTUAL_FLAG | GSS_C_REPLAY_FLAG | GSS_C_DELEG_FLAG | GSS_C_SEQUENCE_FLAG into req_flags).

However, GSS_C_SEQUENCE_FLAG should only be passed in for TCP-based clients, not for UDP-based clients.

3.3 Client Context Establishment Major Status codes

The gss_init_sec_context returned status code can take two different success values:

- If gss_init_sec_context returns GSS_S_CONTINUE_NEEDED, then the client should expect the server to issue a token in the subsequent subnegotiation response. The client must pass the token to another call to gss_init_sec_context, and repeat this procedure until "continue" operations are complete.
- If gss_init_sec_context returns GSS_S_COMPLETE, then the client should respond to the server with any resulting output_token.

If there is no output_token, the client should proceed to send the protected request details, including any required message protection subnegotiation as specified in sections 4 and 5 below.

3.4 Client initial token

The client's GSS-API implementation then typically responds with the resulting output_token which the client sends in a message to the server.

```
+-----+-----+-----+.....+
+ ver  | mtyp | len  |      token      |
+-----+-----+-----+.....+
+ 0x01 | 0x01 | 0x02 | up to 2^16 - 1 octets |
+-----+-----+-----+.....+
```

Where:

- "ver" is the protocol version number, here 1 to represent the first version of the SOCKS/GSS-API protocol
- "mtyp" is the message type, here 1 to represent an authentication message
- "len" is the length of the "token" field in octets
- "token" is the opaque authentication token emitted by GSS-API

3.5 Client GSS-API Initialisation Failure

If, however, the client's GSS-API implementation failed during `gss_init_sec_context`, the client must close its connection to the server.

3.6 Server Context Establishment

For the case where a client successfully sends a token emitted by `gss_init_sec_context()` to the server, the server must pass the client-supplied token to `gss_accept_sec_context` as `input_token`.

When calling `gss_accept_sec_context()` for the first time, the `context_handle` argument is initially set to `GSS_C_NO_CONTEXT`.

For portability, `verifier_cred_handle` is set to `GSS_C_NO_CREDENTIAL` to specify default credentials (for acceptor usage).

If `gss_accept_sec_context` returns `GSS_CONTINUE_NEEDED`, the server should return the generated `output_token` to the client, and subsequently pass the resulting client supplied token to another call to `gss_accept_sec_context`.

If `gss_accept_sec_context` returns `GSS_S_COMPLETE`, then, if an `output_token` is returned, the server should return it to the client.

If no token is returned, a zero length token should be sent by the server to signal to the client that it is ready to receive the client's request.

3.7 Server Reply

In all continue/confirmation cases, the server uses the same message type as for the client -> server interaction.

```
+-----+-----+-----+.....+
+ ver  | mtyp | len  |          token          |
+-----+-----+-----+.....+
+ 0x01 | 0x01 | 0x02 | up to 2^16 - 1 octets |
+-----+-----+-----+.....+
```

3.8 Security Context Failure

If the server refuses the client's connection for any reason (GSS-API authentication failure or otherwise), it will return:

```
+-----+-----+
+ ver  | mtyp |
+-----+-----+
+ 0x01 | 0xff |
+-----+-----+
```

Where:

- "ver" is the protocol version number, here 1 to represent the first version of the SOCKS/GSS-API protocol
- "mtyp" is the message type, here 0xff to represent an abort message

4. GSS-API Protection-level Options

4.1 Message protection

Establishment of a GSS-API security context enables communicating peers to determine which per-message protection services are available to them through the `gss_init_sec_context()` and `gss_accept_sec_context()` `ret_flags` `GSS_C_INTEG_FLAG` and `GSS_C_CONF_FLAG` which respectively indicate message integrity and confidentiality services.

It is necessary to ensure that the message protection applied to the traffic is appropriate to the sensitivity of the data, and the severity of the threats.

4.2 Message Protection Subnegotiation

For TCP and UDP clients and servers, different levels of protection are possible in the SOCKS V5 protocol, so an additional subnegotiation stage is needed to agree the message protection level. After successful completion of this subnegotiation, TCP and UDP clients and servers use GSS-API encapsulation as defined in [section 5.1](#).

After successful establishment of a GSS-API security context, the client's GSS-API implementation sends its required security context protection level to the server. The server then returns the security context protection level which it agrees to - which may or may not take the the client's request into account.

The security context protection level sent by client and server must be one of the following values:

- 1 required per-message integrity
- 2 required per-message integrity and confidentiality
- 3 selective per-message integrity or confidentiality based on local client and server configurations

It is anticipated that most implementations will agree on level 1 or 2 due to the practical difficulties in applying selective controls to messages passed through a socks library.

4.3 Message Protection Subnegotiation Message Format

The security context protection level is sent from client to server and vice versa using the following protected message format:

```
+-----+-----+-----+.....+
+ ver  | mtyp | len  | token |
+-----+-----+-----+.....+
+ 0x01 | 0x02 | 0x02 | up to 2^16 - 1 octets |
+-----+-----+-----+.....+
```

Where:

- "ver" is the protocol version number, here 1 to represent the first version of the SOCKS/GSS-API protocol
- "mtyp" is the message type, here 2 to represent a protection level negotiation message
- "len" is the length of the "token" field in octets

- "token" is the GSS-API encapsulated protection level

4.4 Message Protection Subnegotiation Message Generation

The token is produced by encapsulating an octet containing the required protection level using `gss_seal()/gss_wrap()` with `conf_req` set to `FALSE`. The token is verified using `gss_unseal()/gss_unwrap()`.

If the server's choice of protection level is unacceptable to the client, then the client must close its connection to the server

5. GSS-API Per-message Protection

For TCP and UDP clients and servers, the GSS-API functions for encapsulation and de-encapsulation shall be used by implementations - i.e. `gss_seal()/gss_wrap()`, and `gss_unseal()/gss_unwrap()`.

The default value of quality of protection shall be specified, and the use of `conf_req_flag` shall be as determined by the previous subnegotiation step. If protection level 1 is agreed then `conf_req_flag` MUST always be `FALSE`; if protection level 2 is agreed then `conf_req_flag` MUST always be `TRUE`; and if protection level 3 is agreed then `conf_req` is determined on a per-message basis by client and server using local configuration.

All encapsulated messages are prefixed by the following framing:

```
+-----+-----+-----+.....+
+ ver  | mtyp | len  |          token          |
+-----+-----+-----+.....+
+ 0x01 | 0x03 | 0x02 | up to 2^16 - 1 octets |
+-----+-----+-----+.....+
```

Where:

- "ver" is the protocol version number, here 1 to represent the first version of the SOCKS/GSS-API protocol
- "mtyp" is the message type, here 3 to represent encapsulated user data
- "len" is the length of the "token" field in octets
- "token" is the user data encapsulated by GSS-API

6. GSS-API Security Context Termination

The GSS-API context termination message (emitted by `gss_delete_sec_context`) is not used by this protocol.

When the connection is closed, each peer invokes `gss_delete_sec_context()` passing `GSS_C_NO_BUFFER` into the `output_token` argument.

7. References

[RFC 1508] Linn, J., "Generic Security Service API", September 1993.

[RFC 1509] Wray, J., "Generic Security Service API : C-bindings", September 1993.

[SOCKS V5] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol V5", RFC 1928, April 1996.

8. Acknowledgment

This document builds from a previous memo produced by Marcus Leech (BNR) - whose comments are gratefully acknowledged. It also reflects input from the AFT WG, and comments arising from implementation experience by Xavier Gosselin (IUT Lyons).

9. Security Considerations

The security services provided through the GSS-API are entirely dependent on the effectiveness of the underlying security mechanisms, and the correctness of the implementation of the underlying algorithms and protocols.

The user of a GSS-API service must ensure that the quality of protection provided by the mechanism implementation is consistent with their security policy.

In addition, where negotiation is supported under the GSS-API, constraints on acceptable mechanisms may be imposed to ensure suitability for application to authenticated firewall traversal.

10. Author's Address

P. V. McMahon
ICL Enterprises
Kings House
33 Kings Road
Reading, RG1 3PX
UK

EMail: p.v.mcmahon@rea0803.wins.icl.co.uk
Phone: +44 1734 634882
Fax: +44 1734 855106