Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

To learn the current status of any Internet-Draft, please check the "id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Abstract

This specification describes a means of using the Generic Security Service Application Program Interface (GSS-API) to secure World Wide Web transactions. Use of the GSS-API enables mutual authentication and data encryption capabilities to be incorporated into Web clients (browsers) and servers in a security technology-independent manner. Such an approach facilitates the use of Web applications in a variety of security environments, and provides a basis for Web applications to negotiate common security mechanisms at run-time.

1. Introduction

The role of the GSS-API in facilitating Web security is to provide a standard programming interface that Web applications can use to secure transactions over the Internet, as per the IETF requirements for Web transaction security [1]. While the API itself does not define a network/wire protocol, implementations of the API define standard network security token formats and protocols for secure client/server communications. Like the Secure Sockets Layer (SSL) [2], the GSS-API can be used to secure other applications in addition to the Web; like the Secure HyperText Transfer Protocol (S-HTTP) [3], the GSS-API supports a variety of security mechanisms. Thus, the GSS-API enables use of a uniform security
interface across Web and other electronic commerce applications, and enables deployment of those applications in a variety of security environments.

2. GSS-API OVERVIEW

The Generic Security Service Application Program Interface (GSS-API) has been under development by the IETF Common Authentication Technology (CAT) Working Group (WG) for several years [4][5]. During this time, the GSS-API has evolved to include a comprehensive set of functions for securing client/server applications. This includes functions for credential management, security context establishment, message privacy, and namespace management. The security context functions support both unilateral and mutual authentication. The message privacy functions support both confidentiality and integrity protection of application data.

The GSS-API is "generic" in the sense that it can be implemented using a variety of security mechanisms. As such, the IETF specification for the GSS-API is accompanied by other specifications which define the token formats, network protocols, object identifiers (OIDs), etc., for implementations of the API based on symmetric and asymmetric encryption algorithms [6][7][8].

Public and commercial GSS-API implementations are already available based on Kerberos, DCE, and RSA technology. These implementations incorporate other networking standards as well, including ASN.1, X.509, and FIPS-113.

The GSS-API is continuing to evolve through the efforts of the CAT WG. A mechanism negotiation protocol is being defined to enable GSS-API-based applications to negotiate and select common security mechanisms at run-time [9]. This will facilitate the interoperability of Web applications across different security environments by allowing a Web client and/or server to support multiple GSS-API mechanisms/protocols, whereby a common mechanism can be negotiated and selected per client/server session. Note that such a scenario inherently assumes that the appropriate security credentials are available for the supported mechanisms. Since the mechanism negotiation protocol specification is at an Internet-Draft stage, this document discusses the Web/GSS-API integration both in terms of having the negotiation capability available (i.e. in the future) and not having it available.

In addition to mechanism negotiation, the CAT WG is also defining a corollary API for store-and-forward applications [10]. This will provide uniform security APIs for use across both session-oriented (Web) and store-and-forward applications (such as email), and will facilitate the use of a single set of credentials (single sign-on) for both types of secure applications.
3. WEB APPLICATION INTEGRATION

The GSS-API can be incorporated into a variety of client/server application protocols, including the World Wide Web Hypertext Transfer Protocol (HTTP). The HTTP/GSS-API integration occurs at the network communications level, whereby the initial HTTP connection is strongly authenticated and the entire transaction is protected using GSS-API functions. All types of HTTP transactions are secured, including document retrievals, form submissions, and CGI results.

The GSS-API integration isolates the application protocol and data format (HTML) from the operating characteristics of the underlying security technology (e.g. cryptographic token formats, credential processing, etc.). This approach provides non-intrusive security in that Web transactions are protected (encapsulated) without modifying the standard HTTP/HTML constructs.

3.1 Secure HTTP URL Types

Given the stateless nature of HTTP, secure Web data is typically identified by a unique HTTP URL type (e.g. "https" for SSL, and "shttp" for S-HTTP). That is, since an HTTP session is composed of a single request/response between client and server, it does not avail itself to an initial protocol data unit (PDU) or command sequence being sent (prior to the actual request) to determine whether a server accepts/requires security for the session (as is possible with the WAIS/Z39.50, NNTP, and FTP protocols). While the "Authorization:" MIME header record is available in HTTP, it cannot be used to convey an authentication type simultaneously with a protected request in the case of authentication methods which require a multi-pass handshake to establish a unique session encryption key.

A Web server’s security capability is instead advertised via a secure HTTP URL type on the data being served by that server. The secure HTTP URL type indicates to a client that a particular security method must be used to dereference the URL, and enables the client to then use the appropriate method to secure its request to the server. This is especially important for the secure submission of private form data.

For the Web/GSS-API integration, a new secure HTTP URL type is defined, named "gss-http". This URL type indicates that the associated Web data must be accessed (or submitted) using a GSS-API security mechanism. For GSS-API implementations that support GSS-API mechanism negotiation, a common security mechanism type and mechanism options are selected via the mechanism negotiation protocol upon dereferencing the gss-http URL type. A security context is then established using the selected mechanism.
In the case of Web applications that only support a single GSS-API mechanism, a security context is established upon dereferencing the gss-http URL type provided both the client and server support the given mechanism. Otherwise, the GSS-API security context establishment functions return an error.

(A request is being submitted to IANA for a standard gss-http port number.)

### 3.2 Acquiring Credentials

Before a security context can be established, a Web client and server must each acquire their respective credentials. This is accomplished by either explicitly using the GSS_Acquire_cred() function, or specifying that default credentials should be used. Acquiring credentials makes available the relevant credential data for the client and/or server to be authenticated via the subsequent context establishment. Once acquired, credentials can be re-used until their lifetime expires, at which time the credentials must be re-acquired. The client and server may have different lifetimes for their credentials. For example, client credentials may have a shorter lifetime than server credentials, since the client host environment may be less secure than that of the server.

Given that the underlying credential acquisition process is dependent upon the type of GSS-API mechanism being used, one or more mechanism-specific (non-GSS-API) functions must typically be called prior to acquiring credentials, as specified by the GSS-API implementation provider. For example, the end-user may need to be prompted periodically for a login ID and password prior to acquiring credentials. Example credential data includes Kerberos tickets, private/public key pairs and public key certificates, etc. Note that the process of acquiring credentials may involve network communication with a key management service (e.g. a KDC or CA service).

### 3.3 Security Context Establishment

Once credentials are acquired, a Web client/server security context can be established using the pertinent GSS-API functions. The security context establishment process authenticates the client and server identities, and establishes a secret key for protecting the subsequent Web transaction. In GSS-API parlance, the Web client initiates the security context, and the Web server accepts the security context (i.e. the server is the target).

#### 3.3.1 Initiating a Security Context

The Web client initiates a security context using the GSS_Init_sec_context() function, passing the following as input arguments:
The credential handle returned from `GSS_Acquire_cred()`, or null to use default credentials.

- The name of the server
- A mechanism type (which may be the negotiation mechanism)
- A set of context-quality flags

The server name is constructed from the secure URL being dereferenced. The target server host name included in the URL is used to form a GSS-API host-based service name. For example, given the secure URL "gss-http://www.tradewave.com/", the server name is constructed as "http@www.tradewave.com". The server name is used by the underlying GSS-API mechanism to obtain the appropriate server credential information needed to authenticate the client with the server. This provides a means of directly associating the target URL host name with the server’s identity to be authenticated.

In environments where multiple authentication domains exist (e.g. Kerberos realms and/or public key certification authorities), additional information may be needed to translate the URL into an appropriate server name for authentication (e.g. a directory service lookup to determine the realm/CA in which the server resides). Also in such environments, consideration must be given as to which domains are trusted; however, this issue is outside the scope of this document.

If mechanism negotiation is supported, the mutual/anonymous authentication state flags are set as per the mechanism type and mechanism options selected during the mechanism negotiation phase. That is, the mechanism negotiation process is used to determine whether mutual authentication is required, e.g. based on the server configuration. If mechanism negotiation is not supported, mutual authentication is used by default. It is recommended that the replay detection and message sequencing flags (`replay_det_req_flag` and `sequence_req_flag`, respectively) also be enabled.

If successful, `GSS_Init_sec_context()` returns a context token, which the client then sends to the server. Depending on the GSS-API mechanism being used, the client may subsequently receive another context token back from the server, which the client then passes into a second call to `GSS_Init_sec_context()`. The `GSS_S_CONTINUE_NEEDED` status returned from `GSS_Init_sec_context()` indicates whether the client should expect to receive a context token back from the server. The client/server context token exchange continues until the `GSS_S_COMPLETE` status is returned from `GSS_Init_sec_context()`. The token exchanges occur as per the token protocol described in section 3.6.

If the security context initiation process fails, the client disconnects from the server.
3.3.2 Accepting a Security Context

The Web server accepts a security context using the GSS_Accept_sec_context() function, passing the following as input arguments:

- The credential handle returned from GSS_Acquire_cred(), or null to use default credentials
- The context token received from the client

If successful, GSS_Accept_sec_context() returns the following as output arguments:

- The name of the client user
- A set of context-quality flags
- Message protection service flags

The returned client name represents the authenticated identity of the user. This is an internal name which can be passed to the GSS_Display_name function to obtain an external (human-readable, printable) name, as well as the name type OID. The format of this name is dependent upon the type of GSS-API mechanism being used. The external representation of the authenticated user name can subsequently be used to enforce access controls on client requests. In addition, the external name should be passed to CGI programs in an appropriate environment variable.

The returned context-quality flags indicate the quality parameters associated with the security context, as conveyed by the client. The message protection service flags indicate whether message confidentiality and/or integrity services are available for the context (via subsequent GSS-API function calls).

Depending on the GSS-API mechanism being used, GSS_Accept_sec_context() may return another context token to be sent back to the client. This is indicated by the GSS_S_CONTINUE_NEEDED status being returned from GSS_Accept_sec_context(); otherwise, the GSS_S_COMPLETE status is returned. The client/server context token exchange continues until the GSS_S_COMPLETE status is returned from GSS_Accept_sec_context(). The token exchanges occur as per the token protocol described in section 3.6.

If the security context acceptance process fails, the server disconnects from the client. Example context acceptance failures include detection of a normal (non-secure) HTTP request, an unsuccessful return status from GSS_Accept_sec_context(), etc.

In the case of a Web server that creates sub-processes to handle client requests, it is recommended that the context acceptance
occur in the sub-process rather than the parent server process, as this will increase the efficiency/throughput of the parent.

3.4 Message Protection

After a security context is established, the Web client and server can exchange information in a secure manner using various GSS-API functions. Transactions can be secured by a Message Integrity Code (MIC) alone, or by a MIC with encryption of the transaction data as well. For public key implementations of the GSS-API, the transaction integrity check can be in the form of a digital signature.

The client and server use the GSS_GetMIC() and GSS_VerifyMIC() functions to generate and verify a transaction MIC, respectively. If encryption is also desired, the GSS_Wrap() function is used to generate a MIC and encrypt the transaction, while GSS_Unwrap() is used to verify the MIC and decrypt the transaction. As mentioned previously, these functions do not actually perform any network communication. Rather, GSS_GetMIC() and GSS_VerifyMIC() return/accept a MIC as a token, which is sent/received as per the token protocol described in section 3.6. Likewise, GSS_Wrap() and GSS_Unwrap() return/accept an encapsulation of a given transaction message, which is also sent/received as per the token protocol described in section 3.6.

3.4.1 Quality of Protection

Immediately after a security context is established, the server sends the client a message that is secured using GSS_Wrap() according to the degree of transaction security being enforced by the server (e.g. as per server configuration parameters). The client uses GSS_Unwrap() on this message to determine the level and type of transaction security required by the server. That is, the server indicates to the client whether integrity alone or integrity with encryption is required, and which integrity/encryption algorithm(s) must be used (as well as key size(s) if variable-size keys are supported by the underlying GSS-API mechanism). Note that since GSS_Unwrap() returns this information via function output arguments (conf_state and qop_state), the message content itself is irrelevant with respect to the Web/GSS-API protocol. The client and server then protect all subsequent messages using the same quality of protection as indicated by the server. These subsequent messages are standard HTTP requests and responses. This enables service providers to specify the level of security needed for their particular service, as per the security policies governing the information content being provided by the service.

The level and type of transaction security indicated by the server to the client must be commensurate with that supported by the
underlying GSS-API mechanism. If the mechanism does not support
the desired quality of protection (as detected by a GSS_Wrap() failure), the server deletes the security context and disconnects
from the client. Likewise, if the server indicates a quality of
protection which the client does not support, the client deletes
the security context and disconnects from the server.

When the server receives messages from the client, it verifies that
the messages are protected in the same manner as was originally
indicated by the server. If a different quality of protection is
detected, the server deletes the security context and disconnects
from the client.

3.5 Context Deletion

After a Web client/server transaction is completed, or an
unrecoverable error condition is detected, the security context is
deleted by both the client and server using the
GSS_Delete_sec_context() function. The output_context_token
argument to this function is not used (null), since the client and
server are both aware of when the transaction ends (and thus don’t
need to inform each other of context deletion). In addition, if
the credentials used to establish the security context were
explicitly acquired by GSS_Acquire_cred() and are not to be used
again, those credentials are released using the GSS_Release_cred() function. However, it is recommended that clients and servers
reuse such credentials to establish multiple security contexts over
a finite credential lifetime, then release those credentials upon
expiration of the lifetime. This approach is more efficient since
the credentials are not re-acquired for each client/server security
context.

It may also be possible for clients to cache and reuse the security
context associated with a given target server, if context reuse is
supported by the underlying GSS-API mechanism. This would minimize
the context setup overhead for each connection to the given server.
However, the GSS-API mechanism may be caching the information it
uses to create the security context, in which case the context
setup overhead is already being minimized.

3.6 Web GSS-API Token Protocol

The GSS-API functions do not actually perform any network
communication between the Web client and server. Rather, a GSS-API
function may return a token which is sent by the client to the
server (or vice-versa), which then passes the token into another
GSS-API function for processing. Prior to having a security
context established, the following protocol is used by the client
and server for sending/receiving GSS-API tokens:
tokenLength               4 bytes
tokenLengthChecksum       4 bytes
majorVersionNumber        2 bytes
minorVersionNumber        2 bytes
reserved (future use)     4 bytes
token                     tokenLength bytes

The token length checksum is simply the negative (two’s complement) of the token length. The checksum can be used to validate the token length, as well as to determine if a GSS-API token is being received. For example, a secure Web server can use this to determine if a normal (non-secure) HTTP request is being received, and if so, to return an appropriate error code. The protocol version information can be used to detect/support backwards-compatibility as the protocol evolves.

Once a security context is established between the client and server, the protocol is simplified to the following for sending/receiving all subsequent GSS-API tokens (e.g. secure message tokens):

  tokenLength               4 bytes
  tokenLengthChecksum       4 bytes
  token                     tokenLength bytes

3.7 Client GUI Considerations

It is recommended that the client GUI provide a visual indicator that differentiates secure Web transactions from non-secure transactions. The indicator could be activated while a secure transaction is occurring, and/or after a secure transaction has completed and a secure page is being viewed.

4. SECURITY CONSIDERATIONS

Security issues are discussed throughout this document. Additional issues regarding GSS-API-based Web transactions involving firewall proxy servers or other server intermediaries will be addressed in future versions of this document.

5. REFERENCES


6. ACKNOWLEDGEMENTS

The author thanks his colleagues at TradeWave for their review of earlier drafts of this document, and also thanks Stephen Farrell of Siemens-Nixdorf and John Linn of OpenVision Technologies for their helpful comments.

This work was initiated at TradeWave (formerly EINet) and funded in part by the ARPA Manufacturing Automation and Design Engineering (MADE) program, under contract management by the USAF Wright Laboratory.

7. AUTHOR’S ADDRESS

Doug Rosenthal
TradeWave
3636 Executive Center Dr., Ste. 100
Austin, TX  78731  USA

Phone: +1 512.433.5347
E-mail: rosenthal@tradewave.com