Overview of Recent Developments on Generic Security Services Application Programming Interface

JP Hong, jph3@cec.wustl.edu

Abstract:

With network security measures becoming more and more complicated, it became essential to provide programmers with the capability to write applications that are generic with respect to security. The Common Authentication Technology (CAT) Working Group of Internet Engineering Task Force (IETF) acknowledged this need and developed the Generic Security Service Application Programming Interface (GSS-API). After ten years of experience with GSS-API, the Kitten (GSS-API Next Generation) Working Group was formed to continue this development. This paper looks at the current issues that Kitten faces in improving the GSS-API.

Keywords:

GSS-API, Kitten Working Group, PRF, SPNEGO

Table of Contents

1. Introduction

The development of Generic Security Service Application Programming Interface (GSS-API) was first launched in July, 1991 by the Common Authentication Technology (CAT) Working Group. The first version was released in May of 1993, and the current version of GSS-API is version 2, which was released in 1997. Within the scope of this paper, GSS-API will refer to version 2. Ever since, GSS-API has provided programmers a single framework in which applications using various security technologies can be implemented. However, providing such a simple answer is never easy, and therefore GSS-API needs more work done to support the most recent technologies. Currently, the dominant security mechanism used with GSS-API is Kerberos. Now, the Kitten Work Group is working to improve the current version of GSS-API, and also to produce a specification for the next generation of GSS-API so that it can support more mechanisms seamlessly.

Back to Table of Contents

2. Overview of GSS-API

GSS-API, on its own, does not provide security. It provides generic guidelines so that application developers do not need to tailor their security implementations to a particular platform, security mechanism, type of protection, or transport protocol. Then, security service vendors provide implementations of their security in a form of libraries, so that even if the security implementations need to be changed, the higher level applications need not be rewritten. This allows a program that takes advantage of GSS-API to be more portable as regards network security, and this portability is the most essential aspect of GSS-API. Figure 1 shows where the GSS-API layer lies.

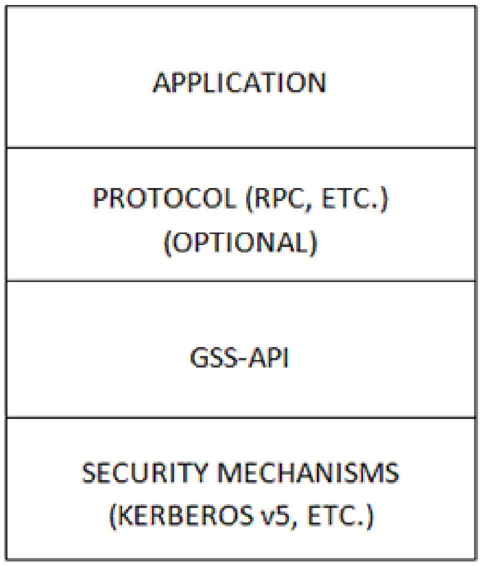
APPLICATION
PROTOCOL (RPC, ETC.) (OPTIONAL)
GSS-API
SECURITY MECHANISMS (KERBEROS v5, ETC.)

Figure 1 – The GSS-API Layer

Two major services that the GSS-API provides are the following[Sun02]:

a. Creation of security contexts that can be passed between applications. Security context is a term used in GSS-API, which refers to a “state of trust” between applications. When two applications share a context, they acquire information about each other, and then can permit data transfer between them while the context is valid. This is shown in stage one of Figure 2.
b. Application of various types of protection, also known as security services, to the data being transferred. This is shown in stage two of Figure 2.

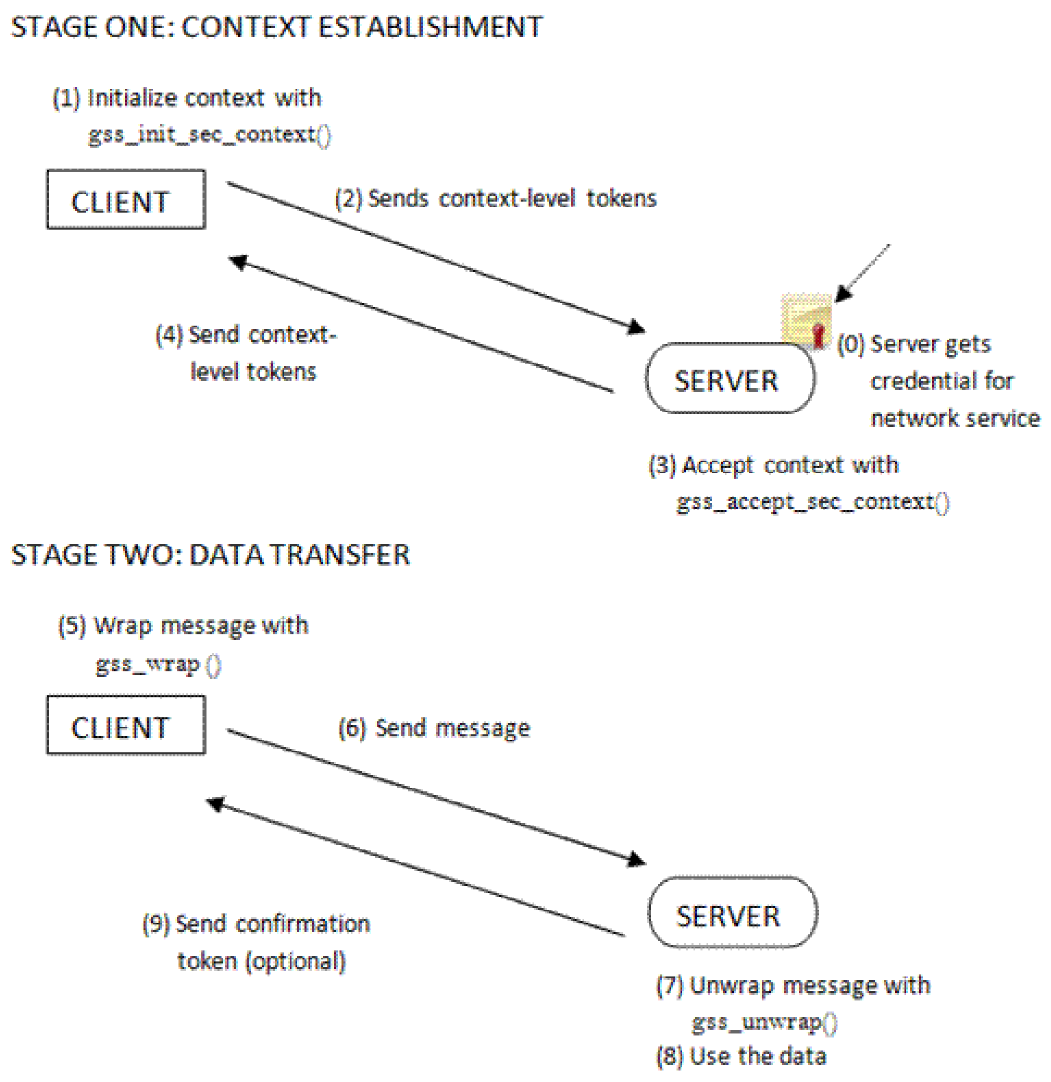
STAGE ONE: CONTEXT ESTABLISHMENT
(1) Initialize context with gss_init_sec_context()
CLIENT
(2) Sends context-level tokens
(4) Send context-level tokens
SERVER
(0) Server gets credential for network service
(3) Accept context with gss_accept_sec_context()
STAGE TWO: DATA TRANSFER
(5) Wrap message with gss_wrap ()
CLIENT
(6) Send message
(9) Send confirmation token (optional)
SERVER
(7) Unwrap message with gss_unwrap()
(8) Use the data

Figure 2 – GSS-API: An Overview

Stage one of Figure 2 depicts the initial context establishment phase of GSS-API. The client sends the host tokens containing the clients' credentials and the types of security mechanisms it supports. The server accepts the context if the information in the tokens match the services it can provide. The function calls gss_init_sec_context() and gss_accept_sec_context() are function calls used in this process. Stage two of Figure 2 shows the actual data being transferred. The appropriate security mechanisms are applied to the data by calling the gss_wrap() function. Likewise, the wrapped data will be changed backed by the gss_unwrap() call and a Message Integrity Code (MIC) will be sent back for integrity check. GSS-API also provides services such as data conversion, error checking, delegation of user privileges, and identity comparison. On the other hand, in order to maximize its generic nature, GSS-API does not provide the following[Sun02]:

a. Security credentials for principals. This must be dealt with by the underlying security mechanisms.
b. Data transfer between applications. This is up to the applications.
c. Ability to distinguish between different types of transmitted data. GSS-API does not determine if a data packet is GSS-API related or not.
d. Status indication of non-regarding GSS-API such as remote errors.
e. Automatic protection of information sent between processes of a multi-process program.
f. Allocation of string buffers to be passed to GSS-API functions.
g. Deallocation of GSS-API data spaces. This must be done explicitly using functions such as gss_release_buffer() and gss_delete_name().

Currently, the GSS-API language bindings are available in C and Java[GSS1]. The Kitten Working Group is working to provide a C# API, however, it has been postponed due to lack of participation.

Back to Table of Contents

3. Recent GSS-API Issues

Recent issues regarding GSS-API focus on keeping up with the latest technologies and addressing how improvements can be made to support more security mechanisms. This section looks into some of these issues.

3.1 Pseudo-Random Function API Extension

Some applications, due to their own reasons, are not able to take advantage of the per-message integrity check (MIC) and token wrapping protection provided by GSS-API. They depend on pseudo-random functions (PRF) to key them in using cryptographic protocols. This brings up the need for GSS-API to be able to key these types of applications. However, the specification of GSS-API does not provide such function, restricting such applications from making use of GSS-API. This need was ackno wledged by the Kitten Working Group, and the PRF API Extension for GSS-API was defined . The PRF API Extension defines a gss_pseudo_random() function that takes as input a context handle, a PRF key, PRF input string, and the desired output length, and outputs the status and PRF output string. The gss_pseudo_random() function must meet the following properties [RFC4401]:

a. The output string must be a pseudo-random function of the input, keyed with the key material derived from the context. The chances of getting the same output given different input parameters should be exponentially small.
b. When applied to the same inputs by two parties using the same security context, both should have same result, even when called multiple times.
c. Authentication must be established prior to PRF computation.
d. It must not be possible to access any raw keys of a security context through gss_pseudo_random().

Pseudo-random functions are said to be capable of producing only limited amounts of cryptographically secure output, and therefore, programmers must limit the use of PRF operations with same inputs to the minimum possible [PRF1]. Also, there is a threat for a Denial of Service attack by tricking applications to input a very long input string and requesting very long output strings. Application developers should therefore place appropriate limits on the size of any input strings received from their peers without integrity protection [RFC4402].

As an extension, this provides an abstract API, and does not address the portability of applications using this extension. In other words, the biggest strength of GSS-API can be sacrificed. The Kitten Working Group is planning to look into this issue in the future development of GSS-API Version 3.

3.2 The Simple and Protected GSS API Negotiation Mechanism

GSS-API provides a generic interface that can be layered on various security mechanisms. If two peers acquire GSS-API credentials for the same security mechanism, that security mechanism will be established between them. GSS-API does not specify a mechanism in which the two peers can agree on a certain mechanism. The Simple and Protected GSS-API Negotiation (SPNEGO) mechanism specifies how this can be done. The steps of the negotiation are as follows [RFC4178]:

a. The context initiator proposes a list of security mechanisms with the most preferred mechanism coming first.
b. The acceptor either chooses initiator’s preferred mechanism or chooses the one that is in the list, and it prefers. If an agreement can’t be made between the two parties, it rejects the list and informs the initiator of this.

The original version of the SPNEGO mechanism was established in 1998. The Kitten Working Group later revised the mechanism and published a new standard, which obsoletes the previous specification. This revision refines the MIC check process of mechanism lists and establishes compatibility with the Microsoft Windows operating systems which implements SPNEGO.

Back to Table of Contents

3.3 Desired Enhancements to GSS API Version 3 Naming

The GSS-API provides a naming architecture that supports name-based authorization. However, with advances in security mechanisms and changes in the way applications that use these mechanisms are implemented, it is required that GSS-API's use of this model to be extended in the following versions. We will look at some of the reasons this change is necessary.

3.3.1 Names in GSS-API

In GSS-API, a name refers to a principal, which represents a person, a machine, or an application. These names are usually in the format of joe@machine or joe@company. In GSS-API these strings are converted into a gss_name_t object by calling the gss_import_name() function. This is called an internal name. However, each gss_name_t structure can contain multiple versions of a single name; one for each mechanism supported by the GSS-API. Calling gss_canonicalize_name(mech_type) on a gss_name_t with multiple versions, will result in a single version of gss_name_t for the specified mechanism. These are called Mechanism Names(MN). This process is shown in figure 3 below.

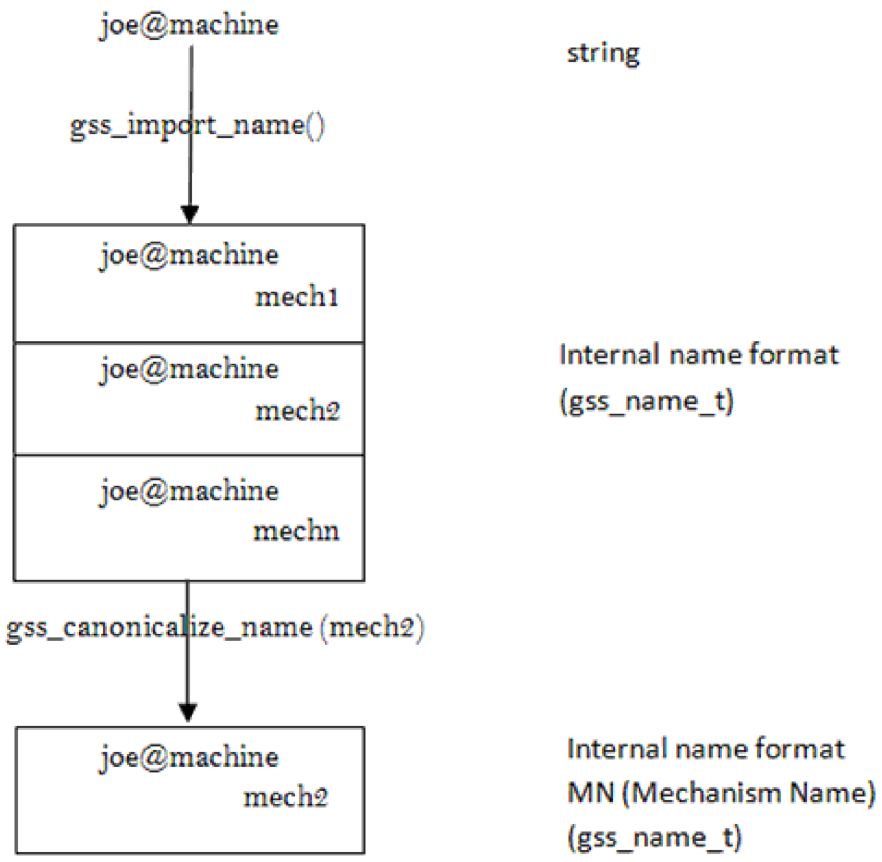

Figure 3 – Internal Names and Mechanism Names

Once converted into MNs, these are stored in an Access Control List (ACL), which contains information on which principals have permission to particular services. The next section looks into the limitations of this architecture and provides some possible methods for enhancement.

Back to Table of Contents

3.3.2 Limitations of Current Names

The GSS-API authenticates two named parties to each other. Once the names of the parties are initially imported into the GSS-API, converted into MNs, and stored in the ACL, then future authorization decisions can be made by simply comparing the name of the requesting party to the list of names stored in the ACL using the memcmp() function as shown in figure 4 [RFC2743, Sun02].

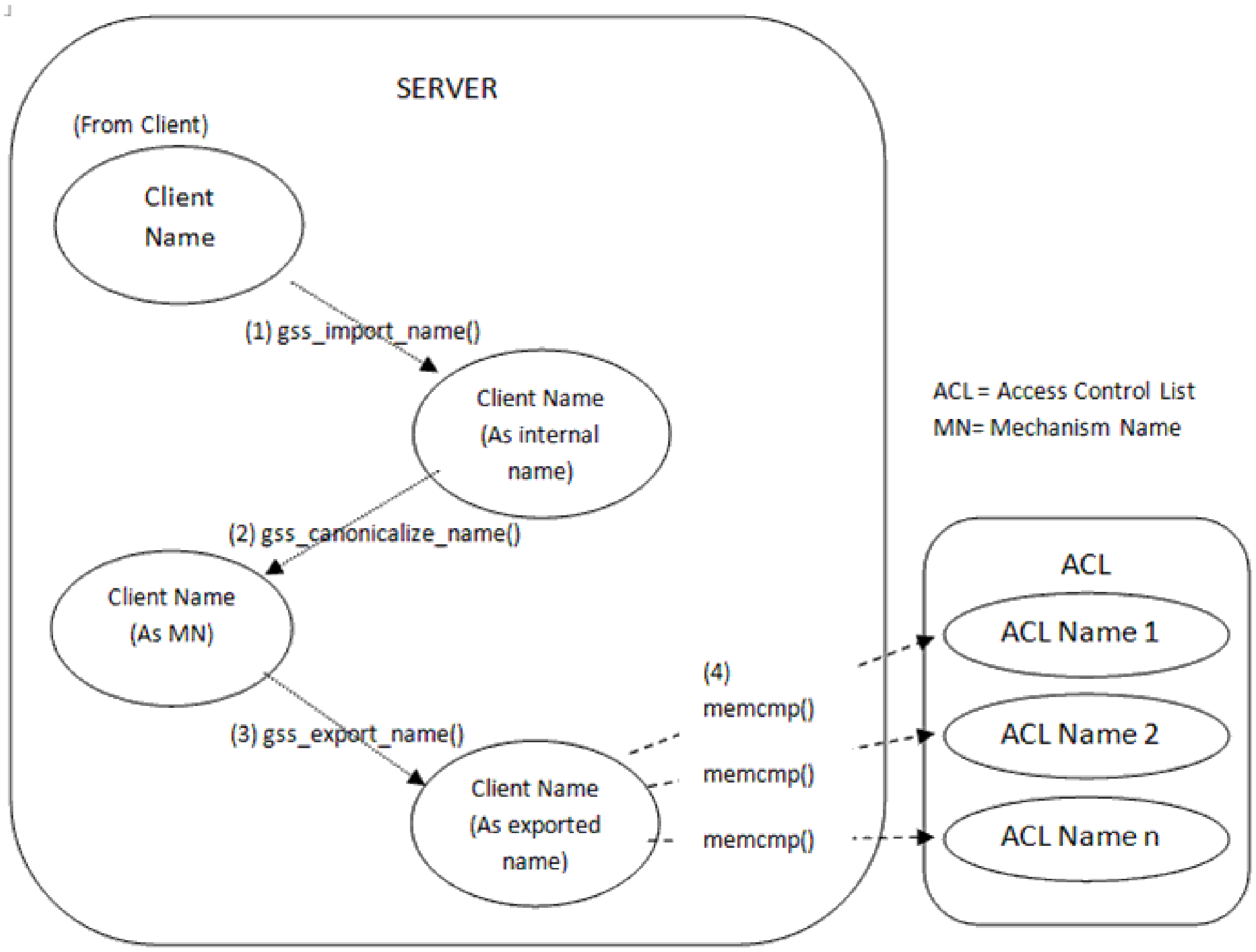

Figure 4 – Comparing Names

However, this can become problematic due to the fact that names can change over time. For instance, if a name contains organization information such as a domain part indicating which department the principal is a part of, this will change when parties move to a different department. Also, if an individual would lawfully change their name, their GSS-API names will change too. The problem lies in the fact that updating ACLs to reflect these changes is difficult [RFC4768]. Another problematic scenario would be when a user account of a machine is deleted. If a new user creates a new account using the same name, the new user will have the privileges intended for the old user.

As GSS-API is used in more complex environments, there are growing desires to use different methods for authorizations such as certificates, Kerberos authorization data, or other non-name-based authorization models. GSS-API's concept of names needs to be enhanced to support these various methods in a mechanism independent manner. The following subsections look in more detail at naming issues regarding the development of new technologies.

3.3.2.1 Kerberos Naming

The Kerberos Referrals document proposes a new type of Kerberos name called an enterprise name [RFC4768]. This addresses the similar problem mentioned above where individuals moving throughout the organization can cause complications. The idea of this enterprise name is to be an alias that the users know for themselves and use to log in to their services. The Key Distribution Center (KDC) will translate this into a principal name and provide the user with tickets authorized to the principal. The enterprise name will track individuals moving throughout the organization. Although the principal name will change for these users, they will not have to change the way they log in since the mapping will be handled in the back end by the KDC.

This enhancement in Kerberos, however, complicates the way GSS-API handles the enterprise names. The future applications implementing Kerberos will ask the users for their enterprise names, requiring a need for GSS-API to handle this transaction. The problem occurs from the fact that Kerberos is not planning to provide a mapping mechanism for translating the enterprise name into a principal name. Thus, any such method would be vendor-specific. It is not possible to implement gss_canonicalize_name for enterprise name types. It will be possible to use the traditional principal names for GSS-API applications, but this will result in losing the benefits of enterprise names. Another issue arising due to enterprise names would be entering these names into the ACL. This would enhance the stability of the ACLs. It seems that this could be accomplished by including the enterprise name in the name exported by gss_export_name. However, this would result in the exported name being changed every time the mapping changes, defeating the purposes of including them in the first place. Kerberos is also looking into mechanisms to include group membership information in Kerberos authorization data. Although it would be favorable to include group names into ACLs, GSS-API currently does not have a mechanism to support this.

3.3.2.2 X.509 Names

X.509 names present a more complicated problem due to the certificates containing multiple options. In most cases, the subject name will be the appropriate name to export in a GSS-API mechanism. However, this field can be left empty in end-entity certificates [RFC 3280, RFC4768]. This leaves the *subjectAltName* extension to be the only portion left to identify the subject. Due to the fact that there may be multiple subjectAltName extensions in a certificate, GSS-API will face the similar problem as it did with group membership in Kerberos. So far, there does not seem to be a sufficient interoperability with GSS-API X.509 mechanisms. Requiring certificates using subject names would limit the mechanism to a subset of certificates. Even with the use of subject names, there is ambiguity in how to handle sorting of name components in GSS-API.

Back to Table of Contents

3.3.3 Possible Solutions

The following subsections look at the possible solutions for the GSS-API Naming Architecture presented by the Kitten Working Group.

3.3.3.1 Composite Names

The first proposal to solve the problems mentioned above would be to extend the GSS-API name to include a set of name attributes. The examples of these attributes are Kerberos enterprise names, group memberships in an authorization infrastructure, Kerberos authorization data, and subjectAltName attributes in a certificate. For this extension to be applied the following operations need to be added[RFC4768]:

a. Add an attribute to a name
b. Query attributes of a name.
c. Query values of an attribute.
d. Delete an attribute from a name.
e. Export a complete composite name and all its attributes for transport between processes.

Most of these attributes will be suitable for storage in a canonical form and binary comparison in the ACL. It is not specified at this point how to deal with the attributes which are not. Due to the various types of attributes, the operation of mapping attributes into names will require a mechanism specific protocol for each mechanism. This solution does come with some security issues. Receiving attributes from different sources may be desirable since the name attributes can carry their own authentication. However, the design to this solution will need to confirm that applications can assign appropriate trust to name components.

3.3.3.2 Mechanism for Export Names

Having a GSS-API mechanism for the sole purpose of having a useful exportable name would be another solution. For instance, this will enable GSS-API to export a name as a local machine user and will work well for name information that can be looked up in directories. The advantage of this solution would be that minimum change is needed to the current GSS-API semantics. However, this is less flexible than the previously stated solution, and is not clear how to handle mechanisms that do not have a wel l-defined name to export, such as X.509.

Back to Table of Contents

4. Summary

As we reviewed in the previous sections, the recent work of the Kitten Working Group have been fixing problems in the current version of GSS-API and designing the future version. GSS-API has so far played an essential role by reducing the burden of dealing directly with security implementations to a minimum for programmers. However, as the newer technologies require more flexibility from GSS-API, the Kitten Working Group will have to make the necessary adjustments in their development of the next generation GSS-API. The continuing challenge for the Kitten Working Group is to provide an enhanced GSS-API that can take advantage of the latest security technologies, and also keep the interface as generic and simple as possible. These two objectives are somewhat contradicting issues. Many security mechanisms are very different, and an attempt to provide a generic interface that supports these various mechanisms does not seem probable. Even in the example of GSS Version 3 naming architecture, we can see this becoming a problem. The currently used naming model takes the names of context initiators and compares them to a set of names on an ACL. The architecture is simple and stable. However, it does not provide the flexibility and features to support the future deployments of GSS-API. The proposed changes for GSS-API Version 3 will increase the complexity of the GSS-API naming architecture. It will have more flexibility to support various security mechanisms, but this also means that there may be more areas vulnerable in regards to security as stated previously. It is to be seen how the Kitten will be able to overcome this dilemma.

Back to Table of Contents

References

[RFC2743] J. Linn “RFC 2743: Generic Security Service Application Program Interface Version 2, Update 1” IETF, Network Working Group, January 2000. http://tools.ietf.org/html/rfc2743

[RFC4178] L. Zhu, P. Leach, K. Jaganathan, W. Ingersoll “The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism” IETF, Network Working Group, October 2005. http://tools.ietf.org/html/rfc4178

[RFC4401] N. Williams “A Pseudo-Random Function (PRF) API Extension for the Generic Security Service Application Program Interface” IETF, Network Working Group, February 2006. http://tools.ietf.org/html/rfc4401

[RFC4402] N. Williams “A Pseudo-Random Function (PRF) for the Kerberos V Generic Security Service Application Program Interface (GSS-API) Mechanism” IETF, Network Working Group, February 2006. http://tools.ietf.org/html/rfc4402

[RFC4768] S. Hartman “Desired Enhancements to Generic Security Services Application Program Interface (GSS-API) Version 3 Naming” IETF, Network Working Group, December 2006. http://tools.ietf.org/html/rfc4768

[RFC3280] R. Housley, W. Polk, W. Ford, D. Solo “Internet X.509 Public Key Infrastructure Certificate Revocation List (CRL) Profile” IETF, Network Working Group, April 2002. http://tools.ietf.org/html/rfc3280

[Sun02] Sun Microsystems, Inc. “GSS-API Programming Guide” May 2002. http://dlc.sun.com/pdf/816-1331/816-1331.pdf

[GSS1] “GSS-API-Wikipedia”, http://en.wikipedia.org/wiki/Generic_Security_Services_Application_Program_Interface

[SPN1] “SPNEGO-Wikipedia”, http://en.wikipedia.org/wiki/SPNEGO

[PRF1] “Pseudorandom function-Wikipedia”, http://en.wikipedia.org/wiki/Pseudorandom_function

[X509] “X.509-Wikipedia”, http://en.wikipedia.org/wiki/X.509

Back to Table of Contents

List of Acronyms

CAT	Common AuthenticationTechnology
IETF	Internet Engineering Task Force
PRF	Pseudo Random Function
GSS-API	Generic Security Service Application Programming Interface
RPC	Remote Procedure Call
MIC	Message Integrity Code
ACL	Access Control List
MN	Mechanism Name
KDC	Key Distributioin Center
SPNEGO	Simple and Protected GSS-API Negotiation

Back to Table of Contents

Last Modified: December, 2007
This paper is available at: http://www.cse.wustl.edu/~jain/index.html