IDfusion
An Open-Architecture for Kerberos based Authorization

Dr. Gregory H. Wettstein, Ph.D., John Grosen, MS.
Information Technology Services
North Dakota State University
Enrique Rodriguez
Safehaus/Apache Software Foundation

4th June 2006

Abstract

Since its initial development Kerberos has evolved to become the widely accepted system for implementing centralized authentication services. During this time the Lightweight Directory Access Protocol (LDAP) has become the accepted method for the centralized distribution of identity information. Organizations increasingly deploy both infra-structural components in order to support management of distributed information delivery systems.

During this evolution no standardized scheme for authorization has emerged. Industry consensus suggests that LDAP is the protocol of choice for storing extended information needed to make authorization decisions. Despite this consensus no standardized scheme has evolved for implementing directory based authorization.

This paper discusses a strategy for using the symmetric key management facilities of Kerberos to implement directory based authorization. The system is architected to provide inherent security in the event of a directory compromise. The system offers the management advantages of role based access systems while providing the option for fine grained authorization control.

The identity based authorization model uses a service oriented approach to managing authorization. As such it is consistent with and supportive of the trend toward services oriented application architectures.

1 Introduction

The Kerberos[7] authentication system is based on a trusted third party authentication model. The Key Distribution Center (KDC) serves as a repository of symmetric keys which are used to cryptographically validate users and the devices or hosts which they access.

The designers of the Kerberos protocol anticipated the potential need for authorization and implemented an optional payload field for carrying authorization information. The format and specification of this field was deliberately left undefined. The only other form of authorization implemented by Kerberos is a system for defining whether or not authentication from foreign realms will be accepted. Currently this is implemented by examining a text file in the user's home directory. This file contains a list of principals which may be used to gain access to the account

Substantial efforts have been conducted to implement authorization on top of Kerberos based authentication. The Secure European System for Applications in a Multi-Vendor Environment (SESAME)[10] implements a role based authorization system using public key based certificates. The Distributed Computing Environment (DCE)[11] also featured extensions for authorization. These systems failed to achieve widespread deployment as strategies for implementing authorization. Both systems require implementation of sophisticated architectures which pose significant entry barriers. In particular a

SESAME implementation involves the deployment of an authorization server at a time when there is a movement to consolidate authentication, authorization and identity management into a common application.

In the time since the development of SESAME and DCE the LDAP[1] protocol has emerged as a standardized scheme for distributing identity information. An LDAP server implements a hierarchical collection of objects which are uniquely named by a Distinguished Name (DN). Each individual object consists of a series of attributes which characterizes the object. While an LDAP directory object has significant flexibility and is an obvious candidate for storing information related to authorization no standardized scheme for doing so exists.

The IDfusion work described in this paper is a strategy for blending the inherent strengths of Kerberos and LDAP to create a system for *implementing* authorization. The goal was to leverage the symmetric key management strengths of Kerberos with the hierarchical information architecture of LDAP directory systems.

The overall design goals for this open-architecture system for authorization are as follows:

- Simple and flexible.
- Synergistic combination of the strengths of Kerberos and LDAP.
- Inherent security from the directory perspective.
- Consistent with services oriented architectures.

Simplicity of design and implementation are considered the foremost goals. In particular a strategy was desired which did not require the implementation of a separate authorization server.

2 Security Considerations and Analysis

The KDC has historically been considered to be a principal component of the Trusted Computing Base (TCB) of an organization. Since it contains authentication secrets for not only users but servers or devices its protection is considered crucial to the security architecture of an organization.

KDC's have been historically hosted on dedicated machines with severely limited access. Organizational security policies may require two man policies for physical access to the machines.

A compromise of the KDC is significant from the perspective of both users and servers/device. A compromise of the KDC places all user authentication secrets in danger thus necessitating users to change their passwords. In addition, all servers or devices which provide Kerberos authenticated services need to have their local key repositories freshened with newly randomized symmetric keys associated with the service principal names.

In order to increase manageability there has been a recent trend to use an LDAP directory server as the database repository for the KDC. Of particular note is Microsoft's Active Directory product[9] which implements both identity information, authorization tokens and authentication secrets in a common data-store. While ease of management is an important goal this practice stands in direct contradiction to well established security practices for KDC's.

If authorization information or directives are implemented in a common directory a compromise of this single resource immediately places the security state of an organization into immediate compromise. An attacker would not only have access to authentication secrets but would be in a position to grant one of the compromised identities access to any of the resources or applications using the directory for authorization information.

This paper advocates an alternate strategy of maintaining separate authentication and authorization stores. A compromise of the authentication store would allow impersonation of a user but would not allow escalation of user privileges beyond those previously granted.

By formulating the authorization identities in the directory using a scheme which involves a cryptographic barrier the utility of a compromised directory is minimized. In our work the cryptographic barrier is provided by using one of the symmetric keys associated with the authentication principal of a service being authorized. Authorization escalation is thus not possible without gaining control of both the directory server and KDC.

3 Fundamental Authorization Concepts

3.1 Implementation vs. Execution

This paper proposes partitioning authorization into two separate components. This partitioning, while not strictly orthogonal, is designed to separate the application specific component of authorization from general processes which can be used by all applications.

Execution of authorization is defined as the application specific process of determining whether or not an authenticated identity should be granted access to a system or resource. By definition execution of authorization is specific to the application. The application may use, for example, attributes within a signed certificate or various attributes of the user such as their organizational role to execute the authorization decision.

In contrast implementation of authorization are the processes or mechanisms by which information needed to execute the authorization decision is made available or conveyed to the application. As noted above this partitioning is not strictly orthogonal. In addition to providing a common framework for conveying application specific information an authorization implementation framework can be leveraged to layer common authorization checks such as general authorization status, time of day or IP address over application specific execution of authorization.

3.2 Authorization Identities

In the IDfusion model the two principal identities involved in implementing authorization are a *user* and a *service.* Users are viewed in the classical sense. A service is considered to be a generic label attached to anything which an organization wishes to authorize access to.

Service labels or identities provide a generic and powerful framework for characterizing authorizations. The well known paradigm of role based access control can be subsumed within this model. In a pure role based system the 'service' can be described as the willingness of the organization to convey a role to the user.

Service labels may have different meanings depending on the application which is requesting authorization to a specific service label. In the reference implementation of IDfusion there is the concept of an EMAIL service. IMAP servers check for EMAIL service rights in order to grant access to IMAP based e-mail accounts. SMTP gateways check for authorization to the EMAIL service to verify that mail should be relayed from a client device. Generically the EMAIL service indicates the desire of the organization to provide access to incoming and outgoing e-mail.

3.3 Identity Intersection

IDfusion models authorization as the genetic intersection or combination of the identity of a user with the identity of a service. The result is a third identity which represents a users right to access a particular service or resource. This third identity is used to represent or label the set of attributes which describes how the user may access the service.

4 Identity Modeling

4.1 Formulation

The previous section discussed authorization as a generic process where the identity of a service is merged or 'fusioned' with the identity of a user to yield a third unique identity. Within the IDfusion model this latter identity represents the willingness of the organization to convey authorization to the user for a specific service. Practical implementation of this system within an LDAP directory requires a substantive definition for these identities.

Each of the three identities is expressed as an N-bit vector. The following terms will be used to define the identities:

1. Uii = User intrinsic identity.
2. Sii = Service intrinsic identity.
3. SIii = Service instance intrinsic identity.

The methodology for creating the N-bit vectors representing the Uii and Sii identities is part of a larger work on secured identity generation from which the IDfusion technology was derived[6]. For the purposes of this discussion the Uii and Sii vectors can be considered to be random vectors guaranteed to be unique within a category, ie. Users and Services.

A cryptographic hash function[2, 3] with message size m=N is used to implement the genetic combination or *fusioning* of the user and service identity. The formulation of the authorization identity is thus expressed as follows:

Algorithm 1 Service Instance Intrinsic Identity

$$SI = Hm(Uii, Sii)$$

The reference implementation currently uses the SHA1 hash[4] as the fusioning algorithm.

4.2 Publication

The Sii and SIii identities are the primary objects published in the directory. The SIii object is published using hexadecimal representation of the numeric hash value as the terminal component of the Distinguished Name (DN) of the object. The following regular expression notation is used to represent the numeric value of the hash:

[0-9a-f]{Lm}

Where Lm is the length of the ASCII representation of the hash value. For a hash function with m = 160, Lm = 40. Using this representation the DN for a service instance identity would be as follows:

SIii=[0-9a-f]{Lm}, dc=something, dc=com

The identity object for the service would be published as follows:

service=SVCNAME, dc=something, dc=com

The rational for not using the Sii value in the DN is to avoid disclosing the value of the service identity.

The Uii identity object is actually published as an SIii object since the publication of user identity information is considered to be a service. A hypothetical organization with one user and one service would publish three objects to support controlling access by the user to the service.

The presence or absence of the SIii object in the directory conveys the willingness of the organization to grant or not grant authorization to whatever the SIii represents. Since the SIii object may contain attributes it serves as a container for storing additional policy information which the application can use in executing the authorization decision.

As an example of this and to demonstrate the importance of the conceptual predicate of the model consider the following three directory objects:

service=SOMESERVICE,dc=something,dc=com
state: enabled|disabled
user=[0-9a-f]{Lm},dc=something,dc=com
state: enabled|disabled
SIii=[0-9a-f]{Lm},dc=something,dc=com
state: enabled|disabled

The first object represents the identity of a service granted by the organization to users. The second object represents the identity of a user while the third object represents the service instance identity granting a user rights to access the service named SOMESERVICE.

An attribute called state has been introduced into the schema definition for the objects. It has two basic values of enabled or disabled. The authorization API can query the attribute value in any of the three objects to determine whether or not the authorization policy decision should be made. Authorization for the service can be turned off in one of three ways:

1. Set the service identity (Sii) to be disabled.
2. Set the user identity (Uii) to be disabled.
3. Set the service instance identity (SIii) to be disabled.

This example underscores the power and flexibility of the underlying authorization model. Choosing option 1 allows an entire category of service to be halted. Option 2 allows disabling all services for a given individual while option 3 discontinues only a specific service for the user.

5 Implementation with Kerberos

5.1 Modifications to Identity Modeling

The identity model discussed in the previous section, while flexible, does not impart inherent security into the directory in the event of a directory compromise. If the Uii and Sii contain the numeric representations of the identities it is a straight forward procedure to compute the SIii identity values and place them in the directory thus granting authorization rights to a particular service. The symmetric key management facilities of Kerberos are used to impart a cryptographic impediment to this process.

The notion of a service having its own unique identity is extended by granting each services its own authentication

identity or principal in the KDC. The following naming scheme is used for these principals:

svc/SVCNAME@REALM

Where SVCNAME is the service tag or label given to a service which the organization conveys authorization for.

The important side effect of creating this principal is the generation of one or more symmetric keys to be associated with the principal. One of these keys is chosen which we will refer to as Kn where n is the bit length of the key. The reference implementation uses the des3-hmac-sha1 where Kn = 192.

The des3-hmac-sha1 key is typically used in derived form. The assumption is made that an appropriate key derivation constant has been allocated and used. Kn thus represents the derived value of the primitive key. The symmetric key is used in conjunction with a Hashed Message Authenticating Coding (HMAC)[5] to generate the SIii identities.

Our standard model for generating an authorization identity using a cryptographic hash function is thus modified to use the keyed variant of the underlying hash function. Our Kerberos based model for combining Uii and Sii identities into an SIii identity is thus represented as follows:

Algorithm 2 Keyed Service Instance Identity
$SIii = HmKn(Uii, Sii)$

The previously described directory publication schema is employed. The only modification is the use of HMAC based SIii's.

It should be noted that the Kerberos implementation of the identity model is not dependent on the encryption algorithm of the key selected. The HMAC algorithm simply requires a binary keyblock. The current implementation supports use of symmetric keys from any encryption algorithm supported by the underlying Kerberos implementation.

5.2 Protocol Implementation

The final component of Kerberos integration is to implement application level support for transport of the SIii. The strategy used is to have the authorization payload field of a Kerberos service credential carry the SIii to the application. The authorization support API extracts the identity and uses it to construct a DN for accessing the authorization identity object in the directory.

Presence or absence of the object in the directory is used to indicate whether or not authorization rights have been granted to the user. Attributes attached to SIii object can be further queried to determine whether or not authorization is disabled or enabled, optionally, the attributes can be supplied to the application which can execute a more sophisticated authorization decision.

To implement this system the AS_REQ (Authentication Service REQuest) and TGS_REQ (Ticket Granting Service REQuest) functions of a KDC are modified to support authorization payload injection. In addition the KDC must be modified to support communications with the directory server containing the user identity and authorization information.

When an AS_REQ is received the principal of the user requesting authentication is used to retrieve the Uii from the directory. The attributes of the user identity object are examined to determine whether or not the user identity has been disabled. If the user identity is active the Ticket Granting Ticket (TGT) is loaded with the Uii and returned to the client.

The TGS_REQ functionality of the KDC is modified to examine the authorization payload field of the TGT presented as authentication for a service ticket request. If a Uii payload field is found and the request is for a principal of the form SVC/SVCNAME an SIii is created based on the Uii payload, the Sii for SVCNAME retrieved from the directory server and one of the symmetric encryption keys of the service authentication principal. The presence of the SIii object is then verified in the directory.

In the process the attributes of the Sii and SIii objects are examined to determine if both objects are enabled. If both identity components are active the authorization payload field of the service ticket is loaded with the SIii and returned to the client for subsequent presentation to an application.

6 Reference Implementation

6.1 Overview of Components

A complete reference implementation of IDfusion has been carried out using version 1.4.3 of the MIT Kerberos distribution and version 2.2.20 of OpenLDAP. Application support is provided by an authorization API library called KerDAP. A Pluggable Authentication Module (PAM)[8] has been implemented based on the API services in KerDAP.

In addition a sample client/server application pair was developed using the example applications provided with the MIT source distribution. Checking and validation of the service ticket payload fields was added to the sample application using support functions from the KerDAP library.

6.2 KDC Extensions

Implementing IDfusion requires tight integration between a KDC and directory services. In addition to establishing secured communications between the two services the basic functionality of the KDC needs to be augmented. As noted in the section on protocol implementation AS_REQ and TGS_REQ request processing must be updated to handle authorization payload injection of Uii and SIii's respectively.

One of the design goals was to implement the enhanced functionality in a manner which minimized long term impact on the source code base. To achieve this goal a pluggable extension system was designed for the MIT Kerberos distribution. The generic plug-in system was designed to allow core functioning of the KDC to be overridden using functions provided through dynamically loaded shared libraries. Support for IDfusion was then coded in the form of a shared library which used the general plug-in facility to supply the enhanced functionality.

The following service intervention points are currently defined for the KDC plug-in:

1. init
2. destroy
3. as_req
4. as_req_authz
5. tgs_req
6. tgs_req_authz

The init method slot is called when the shared library plug-in system is initialized. The entry in the destroy function slot is called when the KDC terminates. Its purpose is to cleanup any memory allocations or persistent state which needs to be maintained by the plug-in.

Two options are supplied for modifying the behavior of AS_REQ and TGS_REQ functionality. The as_req and tgs_req slots provide methods for completely overriding the default AS_REQ and TGS_REQ routines. As a more minimal alternative the as_req_authz and tgs_req_authz slots allow a method of overriding only the management of the authorization data payload fields during AS_REQ and TGS_REQ handling.

The shared library plug-in may choose to initialize any of the method slots. A null pointer placed into a method slot is used to signal the generic plug-in system that a method is not being supplied for an intervention point.

A standardized return scheme is used by a plug-in to signal the generic service system on how to handle execution after a plug-in method completes. The following return codes are defined:

1. KRB_PLUGIN_DECLINED
2. KRB_PLUGIN_PROCESSED
3. KRB_PLUGIN_OVERRIDE
4. KRB_PLUGIN_ERROR

The first return code indicates the plugin declined to do anything and standard execution should continue. The second return code indicates the plug-in carried out some action successfully but standard processing should still occur. The override return code is used to indicate the plug-in has successfully executed and its results should be used instead of the standard processing path.

The fourth return value indicates the plug-in encountered an error which should be reported as a return code the replaced functionality. A state structure is maintained by the generic plug-in code which allows the plug-in to provide not only a standard krb5_error_code but a character string which further describes the error.

GSSAPI authenticated and protected connections are used between the KDC and the OpenLDAP directory server. The init method is used by the IDfusion plug-in to initialize credentials and setup the protected connection to the directory server. The library self-generates the necessary service credentials by looking up the ldap/hostname@REALM principal and using the associated key to construct a service ticket.

The destroy slot is used to close the directory connection, destroy the authentication credentials and release all state associated with IDfusion plug-in.

The injection of the Uii intrinsic identity into TGT's was handled by providing a method for the as_req slot. This could have alternately been done through the use of the as_req_authz method slot. The former was chosen since the plug-in also implements identity translation. This latter functionality was provided to support larger goals of the parent project.

The lower impact entry point was chosen for providing the necessary functionality for TGS_REQ handling. The tgs_req slot was filled with a function which interprets the authorization payload field of the TGT, validates the service request and loads the SIii identity into the payload of the requested service ticket.

6.3 KADMIND extensions.

While not strictly applicable to the IDfusion work the generic MIT plug-in infrastructure has been extended to the administrative server as well. This allows KDC management functionality to be enhanced and extended as well.

The current plug-in implementation has the following two method slots defined:

1. pwd_update
2. acl_check

The first slot provides a method for intercepting password change requests. Interception is provided at the level where the password is added to the database. This allows password change requests to be intercepted which come through both the standard administrative protocol as well as the password changing protocol.

The current plug-in implements the storage of the plaintext passwords on the TL_DATA chains associated with each principal. The passwords are encrypted using the master key of the database. This functionality, while not used by IDfusion, supports goals of the larger project.

The second slot is used to override the standard access control checks which the administrative server imposes upon the principal used to authenticate the administrative session. Currently access controls are implemented in the standard distribution through a text file which contains a list of principals with administrative access rights and optional data fields which define what actions the principal may execute.

The current plug-in simply logs the administrative principal, the target principal and the requested action. Work is currently ongoing to use this infrastructure to implement IDfusion based authorization for Kerberos administrative access.

6.4 Service and Authorization Management.

6.4.1 Identity and Service Management Engine (ISME)

Previous discussions in this paper have indicated the need to have integrated management of Kerberos and LDAP based directory services. The Identity and Services Management Engine (ISME) was implemented to provide this integrated management.

ISME provides a Java based application for managing Kerberos authentication principals and the publication of identity information in an LDAP based directory. With respect to IDfusion the primary function of ISME is to provide a system for computing SIii identities and publishing the identities and any related attributes into an LDAP based directory.

ISME manages the identities in an organization using a hierarchical tree structure. An organization can be broken into logical elements for the purposes of managing identities. The application manages the creation of objects within these containers which represent users (Uii's), services (Sii's) and authorization instance identities (SIii's).

The concept of *'binding'* is supported by ISME. The services provided by the organization to users within a management container is defined by the service identities attached to the container. After the services are defined

additional objects defining individual authorizations can be created for users defined in the container.

Binding a service to a management container is translated by ISME into actions which result in the creation of a service authentication principal for the service. As part of this process a symmetric key associated with the principal is extracted from the Kerberos system to be used in creating the authorization identities.

From a security perspective the only two components with access to the authentication keys for the service identity are ISME and the Kerberos server itself. ISME maintains its copies of the key in an encrypted keystore. ISME application startup requires the entry of an administrative password which is used to decrypt the keys and load them into the application.

Since ISME is not an active entity in the authentication/authorization decision process there is no need to implement a local key stash as is commonly done with KDC's. General security of the keys can be enforced by aggressive management of the password required for ISME application startup.

Actions to manipulate user, service and authorization identities are translated into service provisioning actions through Java based plug-ins. The plug-in translates actions into XML encoded directives which are passed into the Service Provisioning Layer (SPL) for interpretation and physical action.

6.4.2 Graphical Object Oriented Interface (GOOI)

Paired with ISME is a client tool for controlling the identity and service provisioning actions. GOOI is currently implemented in Java and provides a graphical representation of the identity and service management tree.

User commands are translated into XML encoded actions to be interpreted by ISME. A GSSAPI authenticated and protected link is used by GOOI to transmit the XML directives to ISME.

Access at the user level to ISME is currently controlled by IDfusion itself. The system defines an ISME_ADMIN service identity which is used to define access to administrative functionality.

6.5 Integration with AFS.

The current reference implementation implements layered authorization for controlling access to AFS file services. This control is separate and orthogonal from the ProTection Services (PTS) implemented in AFS itself.

As noted in the section describing KDC extensions payload injection into service tickets is implemented by providing a fulfillment method via the tgs_req_authz slot. Authorization for AFS services was implemented by implementing a method for the tgs_req slot.

The method checks for the name of the service principal which is being requested. If the request is for an AFS principal (afs@REALM) the method carries out the authorization process to determine if the identity requesting the service has a valid SIii and that the service should be in effect. If not the KRB_PLUGIN_ERROR code is returned from the plug-in. The protocol specific return code is sent to cause an unknown service message to be issued to the client.

From the perspective of the client the failure in authorization is indistinguishable in effect from a failure in authentication. The inability to obtain the correct service ticket effectively bars access to file services.

7 Discussion

The reference implementation demonstrates the feasibility and general utility of an open-architecture system for implementing authorization. The general goals of the system as earlier outlined are fulfilled.

The system provides a very simplistic means of implementing infrastructure needed to support execution of an authorization decision. Authorization for a service is conveyed by the simple action of publishing an object in the directory. This same directory object offers an extensible platform for conveying additional information which may be used by an application specific process to execute a more advanced authorization decision.

Separating the implementation and execution of authorization provides the added advantage of allowing generic authorization decisions to be layered over the top of application specific actions. The authorization access API provides a system whereby an early authorization decision can be imposed without participation by the appli-

cation itself. The only necessary requirement is for the application to ask the authentication question. The success in overlaying an authorization decision onto AFS file services demonstrates the general utility of the strategy.

IDfusion provides a synergistic blending of the functionality of Kerberos and LDAP directory servers to implement an authorization system which is secure in the face of directory compromise. Publication of an authorization identity object requires access to the symmetric key maintained for the service authentication identity by a KDC. Without knowledge of this key a compromise of the directory provides insufficient information for an attacker to insert an identity into the directory to convey additional privileges.

The use of the authorization identity in the authorization payload field of the Kerberos ticket implements a very efficient method for conveying the authorization decision. Unlike other methods which load the field with signed certificates or XML based data there is minimal additional overhead added to either the ticket granting or service tickets.

If a desire for greater security such as the need to prevent privilege escalation of an existing authorization is desired the IDfusion method is compatible with the use of organizationally signed certificates. The concept of the authorization simply being a pointer to a directory object allows arbitrarily complex strategies using larger amounts of information.

Implementing authorization as a directory identity object removes the need to implement a separate authorization server. In effect the directory server itself serves as the authorization server.

The overall objectives are achieved in a manner which benefits from and preserves the isolation of identity information and authentication secrets. This strategy supports sound and well understood field security practices.

8 Future Work

While the work presented in this paper demonstrates the general utility of the practice there are additional opportunities for further work.

8.1 Host ticket transport of authorization identities.

The current model of requiring a service ticket of the form svc/SERVICENAME@REALM requires the application to have access to the key used to encrypt the service principal. A compromise of an application server in conjunction with a compromise of the directory server would result in a loss of authorization security. This risk could be mitigated by using a key for HMAC based authorization identity generation algorithm which is different than the key used for encrypting the service ticket.

This strategy still presents a management issue since the svc/SERVICENAME keys need to be propagated to the hosts delivering the application. ISME currently supports the notion of binding servers to an application. The service plug-in translates this binding into the transport of the required keys to the target server.

An alternative strategy would be to leverage host based service tickets to transport the SIii. This would require transmitting the name of the service being requested in addition to the name of the host for which the service key is being requested.

8.2 Two factor authentication.

Initial design work in two factor authentication continues to demonstrate the general flexibility of the IDfusion model.

The current strategy involves having the user carry their Uii identity with them in a USB based flash disk. The flash disk contains a copy of the users Uii encrypted with an RSA private key managed by ISME. Since the current implementation uses SHA1 as the identity fusioning algorithm the size of the Uii is consistent with RSA encryption payload sizes.

A modified kinit routine loads the encrypted Uii key into the authorization payload field of the request for a ticket granting ticket. The as_req method checks the payload field for an encrypted Uii and if one is found decrypts the field and loads the Uii into the TGT. Failure to decrypt the Uii or finding that the Uii is not in the directory or not enabled is used as the basis for denying authentication.

This strategy could be further extended for service tickets. In this model the user would be expected to carry

RSA encrypted SIii's which would be loaded into the authorization payload of the service ticket request.

9 Conclusions

IDfusion and the concept of service oriented authorization has been demonstrated in both a reference solution and in practical application. The services oriented approach to information delivery and authorization has been implemented and is part of standard operational practices for six institutions of higher education and K-12 in the State of North Dakota. The approach has proven practical and intuitive to both information services technology staff and the user community.

The separation of authorization into an implementation and execution phase provides the framework for a standardized authorization protocol while preserving application specific authorization decisions. A common framework for implementing authorization provides advantages not only for technology management but for developers who can leverage a common system for rapid integration of their applications into an enterprise service management infra-structure.

The success of the INTERNET has clearly demonstrated the importance and power of standardized protocols. The increasing trend toward Services Oriented Architectures provides a powerful incentive for extending this paradigm to the field of authorization management.

10 Acknowledgments

Thanks to Dr. Thomas Moberg, CIO and Information Technology Services of North Dakota State University for the opportunity to work on and develop this technology. A special thank you to the Office of the President Joseph Chapman for providing monetary support for travel and meeting expenses.

A final and very special note of gratitude goes to my co-author Johannes Christian Grosen. His wisdom, professional advice and personal friendship have been an inspiration to the primary author.

References

[1] W. Yeong, T. Howes, S. Kille; *Lightweight Directory Access Protocol*. IETF RFC 1777, March 1995. http://www.ietf.org/rfc/rfc1777.txt

[2] M.J.B. Robshaw; *MD2, MD4, MD5, SHA and Other Hash Functions*. Technical Report TR-101, version 4.0, RSA Laboratories, 1995.

[3] B. Preneel; *Analysis and Design of Cryptographic Hash Functions*. Ph.D. Thesis, Katholieke University Leuven, 1993.

[4] *Secure Hash Standard*; Federal Information Processing Standards Publication 180-1, April 17, 1995. http://www.itl.nist.gov/fipspubs/fip180-1.htm

[5] H. Krawcyk, M. Bellare, R. Canetti; *HMAC: Keyed-Hashing for Message Authentication*; RFC2104; February, 1997.

[6] Dr. Gregory H. Wettstein, Ph.D., John Grosen, M.S.; *A Linux-based Open Source Middleware Initiative.* Proceedings from the USENIX Atlanta Linux Symposium, October 2000. http://www.ndsu.nodak.edu/kerDAP http://www.hurderos.org/documentation/idfusion-hurd.pdf

[7] J. Kohl, C. Neuman; *The Kerberos Network Authentication Service (V5)*; ISI, September 1993. http://www.ietf.org/rfc/rfc1510.txt http://www.ietf.org/rfc/rfc4120.txt

[8] V. Samar, C. Lai; *Making Login Services Independent of Authentication Technologies*; 3rd ACM Conference on Computer and Communications Security, March, 1996. http://www.sun.com/software/solaris/pam/pam.external.pdf

[9] http://www.microsoft.com/windowsserver2003/technologies/directory

[10] P.V. McMahon; *SESAME V2 public key and authorisation extentions to Kerberos;* Proceedings of the 1995 Symposium on Network and Distributed System Security (SNDSS'95), p. 114, February 16-17, 1995.

[11] S.B. Fairthorne; *Security Extension for DCE 1.1;* OSF DCE RFC 19.