Proxy-Based Authorization and Accounting for Distributed Systems

B. Clifford Neuman
Information Sciences Institute
University of Southern California

Abstract

Despite recent widespread interest in the secure authentication of principals across computer networks there has been considerably less discussion of distributed mechanisms to support authorization and accounting. By generalizing the authentication model to support restricted proxies, both authorization and accounting can be easily supported. This paper presents the proxy model for authorization and shows how the model can be used to support a wide range of authorization and accounting mechanisms. The proxy model strikes a balance between access-control-list and capability-based mechanisms allowing each to be used where appropriate and allowing their use in combination. The paper describes how restricted proxies can be supported using existing authentication methods.

1 Introduction

The problem of authentication across computer networks has received much attention in recent years. Authentication is often only a step in the process of authorization or accounting. The goal is to verify that the individual making a request is authorized to do so, or to guarantee that the correct individual is charged for an operation. Despite the close ties among these problems, little progress has been made in providing secure, widespread, distributed mechanisms for authorization and accounting. To date, authorization and accounting have most often been supported locally by a server, instead of by the use of distributed authorization or accounting services. Such authorization and accounting services will be critical as the network is used more and more for electronic commerce and other applications where clients and servers not previously known to one another must interact. By generalizing the authentication model to support restricted proxies, distributed authorization and accounting can be easily supported.

This paper presents a unified model for authentication, authorization, and accounting that is based on proxies. Section 2 defines the term proxy and briefly describes how proxies can be supported by existing authentication mechanisms. The use of proxies for authorization is demonstrated in Section 3. The proxy model strikes a balance between access-control-list and capability-based mechanisms allowing each to be used where appropriate and allowing their use in combination. Section 4 discusses the necessary features of a distributed accounting service and shows how accounting fits the model. Section 5 discusses related work on distributed authorization and accounting. Integration of the described mechanisms with existing authentication systems is discussed in Section 6, and Section 7 discusses some of the more useful restrictions that can be supported. Section 9 draws conclusions.

2 Restricted proxies

A *proxy* is a token that allows one to operate with the rights and privileges of the principal that granted the proxy. Naturally, it must be possible to verify that a proxy was granted by the principal that it names. This is an authentication problem. In fact a principal with the credentials[1] needed to authenticate itself can often grant a proxy to another principal simply by passing on those credentials.

Implementing proxies in this manner has several shortcomings. First, the proxy can be used by anyone that gets hold of it. This won't always be a problem, but in many cases one should be able to specify the principal that is to act on one's behalf. Second, a proxy is all or nothing. The individual who has been granted the proxy can do anything that the grantor could do on any service to which the original credentials applied.

[1] Credentials consist of an encrypted certificate together with information needed to use the certificate.

Proceedings of the 13th International Conference on Distributed Computing Systems, Pittsburgh, May 1993.

Certificate: $[restrictions, K_{proxy}]_{grantor}$
Proxy-key: K_{proxy}

Figure 1: A restricted proxy

A *restricted proxy* is a proxy that has had conditions placed on its use. A principal possessing authentication or authorization credentials can generate a restricted proxy, a new set of credentials which are more restricted than the original credentials; it is not possible to remove restrictions. It must be possible for the server to which a restricted proxy will be presented (the end-server) to verify that the restrictions have not been tampered with. Among the restrictions that are often specified are that the proxy may only be used by a designated principal, or that the operations that may be performed are to be restricted.

When a principal issues a restricted proxy to another principal, the second principal is authorized to perform all operations for which the first principal is authorized on the server or servers for which the proxy is applicable, subject to any restrictions recorded in the proxy. In the discussion that follows, the *grantor* is the principal on whose behalf a proxy allows access. The *grantee* is the principal designated to act on behalf of the grantor. The *end-server* is the server to which the proxy must be presented to perform an operation.

The implementation of restricted proxies relies on the use of encryption-based authentication of the original grantor of the proxy. Either conventional or public-key cryptography may be used. In this section I describe the implementation at a high level, independent of the authentication mechanism in use. The description assumes that the infrastructure needed to authenticate the original grantor of a proxy is in place and messages required by the underlying authentication protocol (e.g., for key distribution) are omitted for clarity. These details, which are specific to the underlying authentication mechanism, are described in Section 6.

A restricted proxy has two parts: 1) a certificate signed by the grantor establishing the proxy, enumerating any restrictions, and establishing an encryption (or integrity) key[2] to be used by the end-server to verify that the proxy was properly issued to the bearer, and 2) a proxy key, an encryption (or integrity) key corresponding to the key embedded in the certificate, that will be used by the grantee to prove proper possession of the proxy. Figure 1 shows the contents of a restricted proxy; square brackets indicate a signature by the principal indicated in the subscript, or under

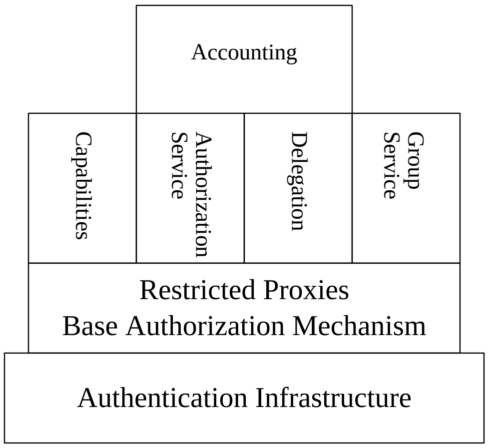


Figure 2: Relationship of security services

a separate encryption (or integrity) key. When a restricted proxy is transferred from the grantor to the grantee, care must be taken to protect the proxy key from disclosure.

There are two classes of proxies: bearer proxies and delegate proxies. A bearer proxy may be used by anyone. A delegate proxy may only be used by a principal named in a list of delegates (encoded as a restriction), or by someone with a suitable additional proxy issued by a named delegate.

To present a bearer proxy to an end-server, the grantee sends the certificate to the server and uses the proxy key to partake in an authentication exchange with the end-server using the underlying authentication mechanism. Usually this exchange involves sending a signed or encrypted timestamp or server challenge, proving possession of the proxy key.

To present a delegate proxy, the grantee sends the certificate to the end-server and then authenticates itself to the end-server under its own identity. The end-server validates the certificate and verifies that the client is included in the list of delegates specified by the proxy.

3 Authorization

Restricted proxies provide the vehicle for implementing a wide range of authorization mechanisms in distributed systems. In this section I describe several such mechanisms and show how they can be supported. Accounting mechanisms are described in Section 4 and build upon the authorization mechanisms described here. Figure 2 shows the relationship of such mechanisms to restricted proxies and to the authentication infrastructure on which they depend.

[2]Depending on the authentication mechanisms in use, this key may require additional protection from disclosure.

Proceedings of the 13th International Conference on Distributed Computing Systems, Pittsburgh, May 1993.

3.1 Capabilities

A capability can be thought of as a bearer proxy that is restricted to limit the operations that can be performed and the objects that can be accessed. No restrictions are placed on the identity of the grantee who is free to pass the capability to others. When presented to the end-server, the grantor's rights (as limited by the restrictions) are available to the bearer.

For example, to create a read capability for a particular file, a user authorized to read that file requests a restricted proxy for use at the file server containing the file (the end-server), but with the restriction that it can only be used to read the named file. The capability is then passed to others who can themselves pass it on. To use a capability, the bearer presents it to the file server in place of (or in addition to) the bearer's own credentials. If the request is to read the file named in the capability, the operation is performed with the rights of the grantor of the proxy.

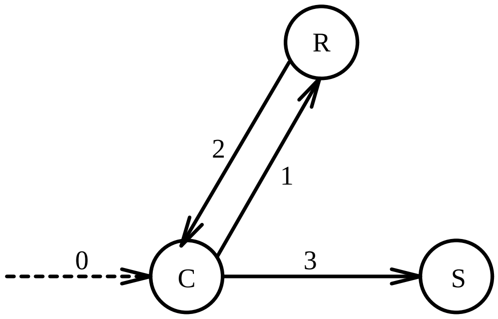
A capability as described above differs from traditional capabilities in several ways. One of the most important distinctions is that in presenting a capability (restricted proxy) to the end-server, the bearer does not send the entire proxy across the network. Instead, the bearer sends the certificate part of the proxy and proves possession by taking part in an authentication exchange using the proxy key as described earlier. The result is that an attacker can not obtain such a capability by tapping the network to observe the presentation of capabilities by legitimate users.

A second distinction is that, as described above, a capability allows a restricted impersonation of the grantor, not direct access to the named object. This means that one can revoke a capability by changing the access rights available to the grantor of the capability. Such a change would affect all capabilities that had been issued by that grantor (as well as any copies), but not those that had been issued by others. If the only principal with *a priori* access to an object is its owner, this distinction disappears as there can be only one original grantor.

A final distinction, as implemented on most authentication systems, is that the resulting capability would have an expiration time. This is a feature. If a non-expiring capability is desired, the expiration time can be set sufficiently far in the future.

3.2 An authorization server

An authorization server implemented using restricted proxies does not directly specify that a particular principal is authorized to use a particular service or access a particular object. Instead, when



1. Authenticated authorization request (operation X)
2. [operation X only]R, {Kproxy}Ksession
3. [operation X only]R, authentication using Kproxy

Figure 3: The authorization protocol

requested by an authorized client, the authorization server grants a restricted proxy allowing the authorized client (the grantee) to act as the authorization server for the purpose of asserting the client's rights to access particular objects. The restrictions in the proxy (in this case a list of authorized actions) are determined by consulting the authorization server's database or applying other suitable heuristics.

Figure 3 shows the messages involved when client **C** uses authorization server **R** for authorization to end-server **S**. The solid lines represent messages in the authorization protocol. The initial request for authorization is authenticated using the underlying authentication protocol. The authorization credentials (a restricted proxy) returned in 2 consist of a certificate and a proxy key. The proxy key is returned protected from disclosure by encrypting it under the session key exchanged during authentication with **R** (encryption is represented by curly braces {}). To use the proxy, the client presents the proxy to the end-server, partaking in an authentication exchange as described in Section 2. Message 0, the dashed line in the figure, represents *a priori* knowledge about the authorization credentials needed for server **S**. This information might be specified as part of the application protocol, retrieved from a name server, or obtained from the end-server directly.

An end-server wishing to use the services of an authorization server would grant full or the maximum desired access to the authorization server (this is described in detail in Section 3.5).

3.3 A group server

A group server implemented using restricted proxies grants proxies that delegate the right to assert membership in a particular group. The protocol is the same as that for the authorization server in figure 3; the authorized operation is the assertion of group membership.

Certificate: $[restrictions1, K_{proxy1}]_{grantor}$
Certificate: $[restrictions2, K_{proxy2}]K_{proxy1}$
Certificate: $[restrictions3, K_{proxy3}]K_{proxy2}$
Proxy-key: K_{proxy3}

Figure 4: Cascaded proxies

A group server might maintain more than one group. The name of a group as asserted by the group server is unique only for a particular group server (or a small set of servers). As such, a global name of a group is composed of the name of the group server, and the name of the group on that server.

It should be possible for the name of a group to appear in authorization databases anywhere that the name of any other principal might appear. This might be on the end-server, or in an authorization server, or even on another group server. An end-server wishing to use a group server would include the name of a group in its authorization database. A client would obtain a group proxy from the group server and send it to the end-server when requesting an operation. The end-server would verify the authenticity of the proxy and the identity of the client, and if valid perform the operation.

If the end-server's authorization database is maintained by an authorization server, then the client would present the group proxy to the authorization server, and if all checks out, the authorization server would return an authorization proxy to be used by the client as described in the previous subsection.

3.4 Cascaded authorization

In a paper on cascaded authentication [11], Sollins proposed a method to pass authorization from party to party when a task involves cascaded operations by parties that do not completely trust one another. A similar mechanism is supported more efficiently by restricted proxies.

By its definition, a proxy allows one principal to perform an operation on behalf of another. An *intermediate server* that has been granted a bearer proxy can pass that proxy to a *subordinate server* (the next server in the pipeline) with additional restrictions applied. Restrictions are added by signing a new proxy with the proxy key from the original proxy. The new proxy specifies any additional restrictions and a new proxy key. The certificates from both proxies are provided to the subordinate server, but only the proxy key from the final proxy in the chain is provided. Figure 4 shows a chain of proxies that might be provided to a subordinate server.

Cascaded authorization is a little different for delegate proxies. To pass a delegate proxy to a subordinate, an intermediate server provides the subordinate with the certificate from the original proxy. Because the intermediate server is explicitly named in the original proxy, it also grants the subordinate a new proxy allowing the subordinate to act as the intermediate server for the purpose of executing the original proxy. Instead of signing the new proxy with the proxy key from the original proxy, it is signed directly by the intermediate server. An important difference between the two approaches to cascaded authorization is that the use of a delegate proxy leaves an audit trail since the new proxy identifies the intermediate server.

A distinct difference between the cascaded authentication approach described by Sollins and the approach described here is that in Sollins's approach the end-server has to contact the authentication server to verify the authenticity of a chain of proxies.

3.5 Access-control-lists and capabilities

By basing authorization on the proxy model, application servers can easily combine the benefits of access-control-lists and capability-based authorization mechanisms. Application servers would be designed to base authorization on a local access-control-list. Where a capability-based approach is required, the access-control-list would contain a single entry naming the principal (perhaps the server itself) authorized to grant capabilities for server operations.

Similarly, when appropriate to hand off the authorization function to a centrally maintained authorization or group server, the name of the authorization or group server would be added to the local access-control-list. In fact, if local autonomy is desired, local users might appear directly in the access-control-list together with the name of an authorization server to which the function of authorizing remote users has been assigned.

Since the same access-control-list abstraction should be used on the authorization servers as on other servers, access-control-list entries can support an associated list of restrictions. On an authorization server, the restrictions field of a matching access-control-list entry can be copied to the restrictions field of the resulting proxy. These would be in addition to restrictions transferred from any proxies presented to the authorization server or those imposed by the server itself.

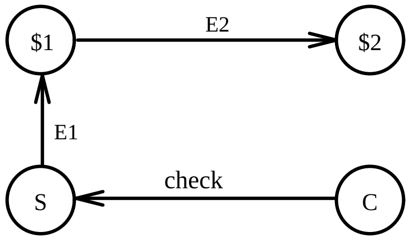
Finally, by supporting compound principal identifiers in access-control-list entries, it becomes possible to require the concurrence of multiple principals for certain operations. Among other things, this functionality allows one to specify the need for both user and host credentials for certain operations as well as the separation of privilege so that a single user can't act alone. Proxy-based authorization allows a user to obtain proxies from more than one grantor for a particular operation, providing the mechanism by which the user would assert such concurrence.

4 Accounting

Section 3 showed how restricted proxies support a wide range of authorization mechanisms. Accounting is closely tied to authorization; in fact, the two are interdependent. Authorization depends on accounting when a server verifies that a client has been allocated sufficient resources (e.g, quota) to perform an operation. Conversely, accounting depends on authorization to control the transfer of resources from one account to another.

In our design, accounts are maintained on accounting servers. At a minimum, each account contains a unique name, an access-control-list, and a collection of records, each record specifying a currency and a balance. Accounting servers support multiple currencies, either monetary (dollars, pounds, or yen) or resource specific (disk blocks, cpu cycles, or printer pages). Quotas are implemented by transferring funds of the appropriate currency out of an account when the resource is allocated and transferring the funds back when the resource is released. Accounts are identified as the composition of the principal identifier for the accounting server and the name of the account on the server. It is possible to transfer resources from an account on one server to one on another.

The transfer of resources can be accomplished through two distinct mechanisms. The simplest mechanism is used when no guarantee is required that sufficient resources exist. A principal authorized to debit an account (the payor) issues a numbered delegate proxy (a check) authorizing the payee to transfer funds from the payor's account to that of the payee. This check limits the resources that can be transferred, and the payee transfers up to that limit. If the payor uses a different accounting server than the payee, the payee grants its own accounting server a cascaded proxy (endorsement) for the check allowing the accounting server to collect the resources on its behalf. Subsequent accounting servers repeat the process until the payor's accounting server is reached. Once a check is paid, the accounting server keeps track of the check number until the expiration time on the check. If, within that period, another check with the same number is seen, it is rejected.



check: [ckno,amount,S]C
E1: [ckno,amount,S]C [dep ckno to $1]S
E2: [ckno,amount,S]C [dep ckno to $1]S [dep ckno to $2]$1

Figure 5: Processing a check

Figure 5 shows the messages involved in issuing and clearing such a check. In the figure, accounting servers are labeled by **$**s. The first message represents a check signed by **C** drawn on **C**'s accounting server **$2** made payable to server **S**. Upon completion of **C**'s request, **S** endorses the check and deposits it with its accounting server in message E1. The endorsement is a restricted proxy that will be used for cascaded authorization. A restricted endorsement (e.g. for deposit only) is a delegate proxy, an unrestricted endorsement is a bearer proxy.

In this case, **C** and **S** do not share the same accounting server, so **$1** marks the resources added to **S**'s account as uncollected, adds its own endorsement and forwards the check to **$2** in message E2. If necessary, such endorsements can be repeated until the check reaches the client's accounting server, but in this case only one additional step is necessary. This distributed method for accounting requires out-of-band mechanisms to deal with checks returned for insufficient resources, or because they are forged or misdrawn, but the same is true in the real world.

The second approach for transferring resources is used when a server requires a guarantee that sufficient resources have been allocated to the client, as is often the case when maintaining quotas. The approach is analogous to that of a certified check. The client draws a check and provides the details (the check number, the party to be paid, and the amount) to the accounting server. The accounting server places a hold on the resources and returns an authorization proxy to the client certifying that the client has sufficient resources to cover the check. The client presents the authorization proxy and the check to the end-server along with its application request.

Once the requested operation is performed, the end-server negotiates the check as described earlier. When the check reaches the client's accounting server, the accounting server looks for the check in its list of outstanding certified checks, and if found, makes the transfer. Cashier's checks are also easily supported by this accounting model; the details are left as an exercise for the reader.

5 Related work

This section describes other work that has been done on authorization and accounting for distributed systems. Some of the earliest work in the area is found in Grapevine [2] where end-servers query registration servers to determine whether a client is a member of a particular group. A similar approach is employed in Sun's Yellow Pages where centrally maintained files such as /etc/group are consulted for authorization purposes. In both approaches, the authorization decision remains with the local system. With the distributed authorization and group services supported by restricted proxies, the authorization decision can be delegated to a remote server.

There have been several proposals concerning forwarding and delegation of authentication in distributed systems. Karger [6] proposed a server that keeps track of special passwords that are established when a user logs in. These passwords are passed to other systems which act on the user's behalf for operations that require the cascaded use of multiple servers. This scheme is not encryption-based, but relies on secure channels for passing the special passwords. These channels might be implemented on top of an end-to-end encryption mechanism.

A mechanism that comes close to restricted proxies is the cascaded authentication mechanism described by Sollins [11] in which restrictions can be added as credentials are passed from system to system. The differences between Sollins' approach and proxy-based cascaded authorization was described in Section 3.4.

The proxy model described by this paper was designed for use in Version 5 of the Kerberos authentication system. Support for proxies was first included in the Kerberos protocol specification in mid 1989 [7]. At about the same time, another mechanisms for delegation was developed as part of the Digital Distributed System Security Architecture [4, 5]. In the DSSA, principals generate and sign delegation certificates to allow intermediate systems to act on their behalf. An important difference is that in the DSSA, restrictions are supported only by creating separate principals, called roles, and by generating a delegation certificate for one of the roles instead of for the original principal. The delegation then supports only access specifically authorized for that role. The creation of a new role is cumbersome when delegating on the fly

Certificate: $\{restrictions, K_{proxy}\}K^{-1}_{grantor}$

Proxy-key: K^{-1}_{proxy}

Figure 6: A public-key restricted proxy

or when granting access to individual objects. Roles can not be used to implement the authorization server described in Section 3.2.

Functionality similar to that of the authorization and group services of Sections 3.2 and 3.3 has been proposed as part of the European Computer Manufacturers Association standard for security in open systems [1]. The ECMA standard defines Privilege Attributed Certificates (PACs) signed by an authority and certifying that the bearer or a named principal possess certain privileges.

Work is underway for the Open Software Foundation's Distributed Computing Environment that uses restricted proxies as supported by Kerberos to pass authorization information. In particular, they have implemented a privilege attribute server that signs certificates asserting a principal's unique identifier and a set of user groups to which the principal belongs. Plans are in place to extend their mechanism to support delegation [3].

Surprisingly little attention has been paid to the issue of accounting in distributed systems. Sentry [9] lays the groundwork for accounting by describing a mechanism that would be co-located with an authentication and authorization server. Although they share a common mechanism, it seems apparent now that there is little to be gained by requiring all three services to be co-located. Like the accounting mechanism described here, Sentry pointed out the need to support multiple currencies.

Amoeba [8] supports a distributed bank server identical in purpose to the accounting server based on restricted proxies. The protocol used by Amoeba's bank server is significantly different, however. In Amoeba, a client must contact the bank and transfer funds into the server's account before it contacts the server. The server will then provide services until the pre-paid funds have been exhausted. Like the mechanism described here, Amoeba supports multiple currencies.

6 Integration with existing systems

It is straightforward to implement restricted proxies using encryption-based authentication mechanisms based on either public-key or conventional cryptography. This section shows how proxies can be implemented with either approach and describes the specific details of their support in Version 5 of the Kerberos authentication system.

Proceedings of the 13th International Conference on Distributed Computing Systems, Pittsburgh, May 1993.

6.1 Public-key cryptography

The certificate for a public-key proxy contains a proxy key generated by the grantor, the expiration time of the proxy, and the restrictions imposed its use. The proxy key embedded in the proxy certificate is a public key from a public/private key pair. The proxy key provided to the grantee is the other key from that pair. All fields are signed by encrypting them with the grantor's private key. Figure 6 shows a proxy generated in this manner. The signed proxy is additionally tagged with the name of the grantor to enable those needing to verify the proxy to select the correct key.

If the authentication system is purely public-key, a public-key digital signature algorithm can be used in place of the encryption system and the encryption step would be replaced by the sealing of the certificate with a cryptographic checksum. If a hybrid authentication system is used, where subsequent keys are from a conventional cryptosystem, then the proxy key is a conventional key generated by the grantor and the proxy key must be additionally encrypted in the public key of the end-server to protect it from disclosure.

The proxy is returned to the grantee. When the grantee presents the proxy to an end-server, the end-server decrypts the proxy using the public key of the grantor (obtained from an authentication/name server), verifies the authenticity of the proxy, accepts additional authentication from the grantee (either personal authentication for a delegate proxy or proof that it knows the proxy key for a bearer proxy), checks the restrictions, and if all checks out, performs the requested operation.

6.2 Restricted proxies in Kerberos

A proxy implemented using an authentication system based on conventional cryptography is identical to that in figure 6 except that the proxy is accompanied by credentials authenticating the grantor to the end-server. The proxy certificate is encrypted using the session key generated by an authentication server, the session key also having been earlier sealed in the credentials. The proxy key is a secret key generated by the grantor. This key is both sealed in the proxy certificate and securely passed to the grantee. The remainder of this section describes the integration of restricted proxies with Kerberos [12], an authentication system based on conventional cryptography developed as part of MIT's Project Athena.

Kerberos credentials are issued by an authentication server and presented by a client to prove its identity to a particular end-server. Credentials consist of two parts: a ticket, and a session key. The ticket contains the name of the authenticated principal and a session key. It is encrypted using the secret key shared by the end-server and the Kerberos server. The session key is never sent across the network in the clear. The session key is returned to the client encrypted in the session key shared by the client and the Kerberos server.

To prove its identity, a client sends the ticket to the end-server along with an authenticator which has been encrypted using the session key. The authenticator proves that the client actually possesses the session key included in the ticket. Without this step an attacker would be able to reuse a ticket that it obtained by eavesdropping on an earlier exchange.

Kerberos has been in use at MIT since Fall of 1986, and it has been used elsewhere since then. Version 5 of Kerberos [7] is the first major revision of the protocol since its original release and contains several new features important for the practical support of restricted proxies. The inclusion of explicit support for proxies in Version 5 makes their use more transparent to applications which have already been modified to use Kerberos.

The Version 5 ticket and authenticator each have a new field called authorization-data. This field consists of an arbitrary number of typed sub-fields, each of which places restrictions on the use of the ticket. The Kerberos protocol does not specify how the sub-fields are to be interpreted except to stress that restrictions must be additive. Each subfield places additional restrictions on the use of credentials, never removing restrictions or granting additional privileges.

When tickets are requested, the requesting principal can specify that restrictions be placed on their use. When new tickets are issued based on existing credentials, restrictions may be added, but not removed. To add restrictions to an existing ticket, a client generates an authenticator specifying a proxy key in the subkey field and specifying additional restrictions in the authorization-data field. The ticket and authenticator are treated as the new proxy and provided with the new proxy key to the grantee. Once obtained, the grantee can use such a proxy the same way it uses any other credentials issued by the authentication system.

6.3 Discussion

Supporting proxies within an authentication mechanism has several advantages. Transparency is one advantage; a second is that the initial authentication of a user can itself be thought of as the granting of a proxy and restrictions can be placed on the credentials based on the characteristics of the initial exchange with the authentication server.

A disadvantage of using conventional cryptography to implement proxies is that each proxy can be used at only a particular end-server. This is offset by implementing proxies within Kerberos itself since it is possible to issue a proxy for the Kerberos "ticket-granting" service. Such a proxy allows the grantee to obtain proxies with identical restrictions for additional end-servers as needed.

7 Common restrictions

The restrictions field of a proxy should be interpreted as a collection of typed subfields, each type corresponding to a different restriction. This section describes several of the more useful restrictions and some that demonstrate the flexibility of the model. Additional restrictions are described in [10]. Neither should be construed as a complete list.

7.1 Grantee

This restriction specifies a list of principals authorized to use a proxy and the number of principals from the list needed to exercise the proxy (usually one). To use such a proxy a principal must present the authentication credentials of a named grantee, or an additional proxy granted by a named grantee, to the end-server along with the proxy. If the **grantee** restriction is missing, the proxy is a bearer proxy and may be used by anyone possessing it. To exercise a bearer proxy the bearer must take part in an authentication exchange proving possession of the proxy key thus preventing an attacker from using a proxy obtained by eavesdropping on the network.

7.2 For-use-by-group

The **for-use-by-group** restriction specifies the list of groups authorized to use a proxy and the number of groups from the list required. To use such a proxy, the bearer presents the proxy along with additional proxies from appropriate group servers. One way to implement separation of privilege is to require assertion of membership in multiple groups with disjoint members.

7.3 Issued-for

The **issued-for** restriction specifies a list of servers authorized to accept the proxy. This restriction is important for public-key proxies which are otherwise verifiable by and exercisable on all servers.

7.4 Quota

The **quota** restriction specifies a currency and a limit. It limits the quantity of a resource that can be consumed or obtained. It will most often be found in a proxy issued by an accounting server.

7.5 Authorized

The **authorized** restriction specifies a complete list of those objects which may be accessed using the rights granted by a proxy and optionally a list of operations that may be performed on each object. This restriction usually appears in proxies used as capabilities. It also appears in proxies returned by an authorization server. There are no constraints on the form of the object names or the list of operations other than that the grantor and the end-server must agree. These fields are to be interpreted by the end-server.

7.6 Group-membership

This restriction specifies that the grantee is a member of only the listed groups. It would be included in a proxy issued by a group server to limit the groups to which one is a member. Without this restriction, the grantee would be considered a member of all groups maintained by the group server granting the proxy.

7.7 Accept-once

The **accept-once** restriction tells an end-server that it is only to accept a proxy one time. This restriction takes an identifier as an argument. Any subsequent proxy from the same grantor bearing the same identifier and received by the end-server within the expiration time of the first proxy is rejected. A real life example of such an identifier is a check number.

7.8 Limit-restriction

Restrictions that are defined only for particular end-servers are sometimes needed. If a proxy can be used on a server to which some restrictions do not apply, those restrictions must be associated with the name of the server to which they do apply. This is accomplished with the **limit-restriction** restriction which takes a list of servers and a list of other restrictions. The restrictions embedded within this restriction will be enforced by the named servers and ignored by others.

7.9 The propagation of restrictions

Authentication, authorization, and group servers accept proxies and issue proxies. If a proxy is issued based upon a proxy that includes restrictions, those restrictions should be passed on to the proxy to be issued. If a restriction is limited (see **limit-restriction**) then the restriction may be left out if it can be guaranteed that the proxy to be issued, and any proxies that might later be derived from it, can not be used for any of the servers listed in the limited restriction.

8 Status

A beta release of Kerberos Version 5 is available. The release includes support for restricted proxies. Information on the Kerberos release is available from `info-kerberos@mit.edu`. Authorization and accounting services built with restricted proxies are being developed at the Information Sciences Institute of the University of Southern California.

9 Discussion and conclusions

The problems of authentication, authorization, and accounting are closely related. By subtly changing the way one thinks about the problems, the similarities become apparent. By extending an authentication system to support restricted proxies, it becomes possible to support flexible distributed authorization and accounting mechanisms. The proxy model strikes a balance between access-control-list and capability-based mechanisms allowing each to be used where appropriate and allowing their use in combination.

This paper has shown how restricted proxies can be supported using existing authentication systems and how they are used for authorization and accounting. The resulting mechanisms scale and appear natural when compared with their analogues in society.

Acknowledgments

I would like to thank Celeste Anderson, Steven Augart, Steve Bellovin, Deborah Estrin, David Keppel, John Kohl, Ed Lazowska, Joe Pato, Karen Sollins, Bill Sommerfeld, Stuart Stubblebine, and Prasad Upasani for discussions of restricted proxies and comments on drafts of this paper.

References

[1] European Computer Manufacturers Association. Security in open systems: Data elements and service definitions, December 1989. Standard ECMA-138.

[2] Andrew D. Birrell, Roy Levin, Roger M. Needham, and Michael D. Schroeder. Grapevine: An exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, April 1982.

[3] Marlena E. Erdos and Joseph N. Pato. Extending the OSF DCE authorization system to support practical delegation. In *Proceedings of the PSRG Workshop on Network and Distributed System Security*, pages 93–100, February 1993.

[4] M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson. The Digital distributed system security architecture. In *Proceedings of the 1989 National Computer Security Conference*, pages 305–319, October 1989.

[5] M. Gasser and E. McDermott. An architecture for practical delegation in a distributed system. In *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, pages 20–30, May 1990.

[6] Paul A. Karger. Authentication and discretionary access control in computer networks. *Computer Networks and ISDN Systems*, 10(1):27–37, 1985.

[7] John T. Kohl and B. Clifford Neuman. The Kerberos network authentication service: Version 5 draft protocol specification. August 1989. Revised November 1989, October 1990, December 1990, June 1991, September 1992, April 1993.

[8] S. J. Mullender and A. S. Tanenbaum. The design of a capability-based distributed operating system. *The Computer Journal*, 29(4):289–299, 1986.

[9] B. Clifford Neuman. Sentry: A discretionary access control server. Bachelor's Thesis, Massachusetts Institute of Technology, June 1985.

[10] B. Clifford Neuman. Proxy-based authorization and accounting for distributed systems. Technical Report 91-02-01, Department of Computer Science and Engineering, University of Washington, March 1991.

[11] Karen R. Sollins. Cascaded authentication. In *Proceedings of the 1988 IEEE Symposium on Research in Security and Privacy*, pages 156–163, April 1988.

[12] J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the Winter 1988 Usenix Conference*, pages 191–201, February 1988.

This research was supported in part by the National Science Foundation (Grant No. CCR-8619663), the Washington Technology Centers, Digital Equipment Corporation, and the Defense Advance Research Projects Agency under NASA Cooperative Agreement NCC-2-539. The views and conclusions contained in this paper are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any of the funding agencies.

The author may be reached at USC/ISI, 4676 Admiralty Way, Marina del Rey, CA 90292-6695, USA. Telephone +1 (310) 822-1511, email bcn@isi.edu.