Proxy-Based Authorization and Accounting for Distributed Systems

B. Clifford Neuman

Department of Computer Science and Engineering
University of Washington, FR-35
Seattle, Washington 98195

bcn@cs.washington.edu

Technical Report 91-02-01
March 1991

Abstract

In recent years there has been much interest in the secure authentication of principals across computer networks. There has been less discussion of distributed mechanisms to support authorization and accounting. Authorization and accounting are more closely related to authentication than most people realize. By generalizing the authentication model to support restricted proxies, both authorization and accounting can be easily supported. This paper shows how to support restricted proxies in an authentication system, presents an appropriate model for authorization and accounting, and describes how they may be easily implemented on top of restricted proxies.

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 end 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, distributed mechanisms for authorization and accounting. To date, authorization and accounting have most often been supported locally by the service, instead of by using distributed authorization or accounting services. Such services are needed if authorization and accounting are to be accomplished between clients and services that are not previously known to one another.

The problems of authentication, authorization, and accounting are more closely related than most people realize. By generalizing the authentication model to support restricted proxies, distributed authorization and accounting can be easily supported.

This research was supported in part by the National Science Foundation (Grant No. CCR-8619663), the Washington Technology Centers, and Digital Equipment Corporation.

This paper presents a unified model for authentication, authorization, and accounting that is based on proxies. Section 2 defines the term proxy, describes how proxies can be supported by an authentication mechanism, and shows how they can be used for authorization. Section 3 discusses the necessary features of a distributed accounting service, and shows how accounting can be thought of as an authorization problem. Section 4 discusses related work in the area of distributed authorization and accounting. Integration of the described mechanisms with existing authentication systems is discussed in Section 5 and Section 6 describes some of the restrictions to be supported. Section 7 draws conclusions.

2 The Proxy Model for Authorization

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 needed for authentication[1] can grant a proxy to another principal simply by passing on those credentials[2].

Implementing proxies as just described leaves a lot to be desired. First, the proxy can be used by anyone who gets his hands on it. This won't always be a problem, but it would be nice if one could 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 the service to which the original credentials applied.

A *restricted proxy* is a proxy that has had conditions placed on its use. Among the conditions that will often be specified are that the proxy may only be used by a designated principal, and that the operations that may be performed by that principal are to be restricted.

To support restricted proxies, an authentication mechanism must allow credentials to be created which allow a second principal to act on a principal's behalf, and it must be possible to include information within these credentials that places restrictions on their use. The server to which the credentials will be presented (the end-server) must be able to verify that the field has not been tampered with.

It must also be possible for any principal possessing credentials to obtain a new set of credentials which are more restricted than the original credentials; it should not be possible to remove restrictions.

Restricted proxies may be supported by authentication mechanisms based on both symmetric (conventional) and asymmetric (public-key) cryptography. The integration of restricted proxies with these mechanisms is described in Section 5.

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

[2] This is not the case in Kerberos where credentials typically contain the address of the host from which they may be used. To grant a proxy a principal must request (from the Kerberos server) a new set of credentials with a different network address.

Restricted proxies provide the vehicle for implementing a large number of authorization mechanisms in distributed systems. In the rest of this section I will describe several such mechanisms and show how they can be supported. In the following discussion, 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.

2.1 Capabilities

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

For example, to create a read capability for a particular file, a user authorized to read that file would request a restricted proxy to be used at the file server containing that file (the end-server), but with the restriction that it can only be used to read the named file. The capability could then be passed to others who could themselves pass it on. To use a capability, the bearer would present[3] it to the file server in place of (or in addition to) the bearer's own credentials. If the request was to read the file named in the capability, that operation would be performed with the rights of the grantor of the proxy.

A capability as described above differs from traditional capabilities in two important ways. First, 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. The second difference is that, for some authentication systems, the resulting capability would have an expiration date. This is a feature. If one really wanted a capability that wouldn't expire, one could set the expiration date sufficiently far in the future.

2.2 An Authorization Service

To implement an authorization service on top of restricted proxies, one must make a subtle change in the way one thinks about such a service. An authorization service need not specify that a particular principal is authorized to use a particular service, or access a particular object. Instead, it can issue a proxy allowing the grantee to access an object using the rights of the authorization server itself. The proxy would be restricted to allow access to only those objects, and with only those rights, to which the principal which requested the proxy is authorized access.

An end-server wishing to use the services of an authorization server would grant full (or the maximum desired) access to that authorization server (or servers). A client wishing to use the end-server

[3] By present I mean take part in an authentication exchange using the credentials.

would contact the appropriate authorization server, request the necessary proxy, and forward it to the end-server along with credentials proving its own identity.

2.3 A Group Service

As with the authorization server, a slight change in the way one thinks of the problem allows a group server to be implemented on top of restricted proxies. One would normally think of the function of a group server as identifying the groups to which a principal belongs, or the principals that belong to a particular group. Instead, one should think of a group as any other principal. The group would have certain rights defined by its inclusion in access control lists scattered throughout the system. The function of a group server would be to issue proxies delegating those rights to members of the group.

A group server may maintain multiple groups. Further, the names of groups are unique only within a particular group server. As such, the global name of a group is really 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, it could be in an authorization server, or even in another group server.

The end-server would use a group server simply by including the name of a group in its authorization database. If the database was locally maintained, 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 must request the group proxy to be used on the authorization server. The proxy would be passed to the authorization server, and if all checks out, the authorization server would return an authorization proxy which could be used by the client as described in the previous subsection.

2.4 Cascaded Authorization

In a paper on cascaded authentication[8], 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 can be supported by restricted proxies, though some of the tradeoffs are different.

By its definition, a proxy allows one principal to perform an operation on behalf of another. An *intermediate server* that has been granted a proxy to perform a particular operation can pass that proxy on to a *subordinate server* (the next server in the pipeline). Since the proxy that is to be passed on might specify the name of the intermediate server, the intermediate server must also generate its own proxy granting the subordinate server the rights to act on its behalf. The new proxy would include a restriction that it can only be used in conjunction with the original proxy

as well as any other restrictions the intermediate server chooses to apply. At the next stage the process would repeat. The chain of proxies would get longer along with the list of restrictions.

A distinct difference between cascaded authorization based on restricted proxies, and that described by Sollins, is that when using restricted proxies, the end-server does not have to contact the authentication server to verify the authenticity of the chain of proxies. On the other hand, as implemented for Kerberos, the authentication server is involved each time a new proxy is generated (possibly at each stage in the pipeline). In any case, the end-server requires certificates for each principal involved. Depending on the authentication mechanism, they might be obtained from a name server, or the certificates may be forwarded along with the proxies.

Requiring that the authentication server is involved each time a new proxy is generated is not as bad as it seems. In many cases an intermediate server will use the same subordinate server over and over. If this is the case, the intermediate server might consider granting a single restricted proxy to the subordinate server instead of one for each request. Such a *standing proxy* would still require that the intermediate server pass subsequent credentials to the subordinate server[4], but it would eliminate the need to contact the authentication server.

3 The Authorization Model for Accounting

I have shown how authorization can be supported by restricted proxies, but what about accounting? Accounting is closely tied to authorization. In fact, the two are interdependent. Authorization depends on accounting when a server wants to verify that a client can pay for an operation before it is performed. On the other hand, authorization is required before funds can be transferred from one account to another.

Before I talk about network mechanisms to support accounting, let us set down the requirements for the accounting servers themselves. Accounts will be maintained on accounting servers. At a minimum, each account must contain a unique name, an access control list, and a collection of records, each record specifying a currency and a balance. It is important that accounts support multiple currencies. Currencies may be monetary (as in dollars or pounds) or they may be logical currencies such as disk blocks or printer pages. Quotas may be 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 will be identified as the composition of the principal identifier for the accounting server and the name of the account on the server. It must be possible to transfer funds from an account on one server to one on another.

The transfer of funds can be accomplished through two distinct mechanisms. The simplest mechanism is used when no guarantee is required that sufficient funds exist. A principal authorized to debit an account (the payor) issues a proxy (check) allowing the payee to transfer funds from the payor's account to that of the payee. The check would limit the amount of funds that could be

[4]This is required because the proxy can't be used without the session key from the credentials.

transferred, and the payee could transfer up to that amount. If the payor uses a different accounting server than the payee, the payee would grant its own accounting server a cascaded proxy (endorsement) for the check allowing the accounting server to collect the funds for it. The accounting server would repeat the process until the payor's bank has been reached. Once a check has been paid, the accounting server must keep track of the check number until the expiration date on the check. If, within that period, another check with the same number is seen, it will be rejected.

The above method of transferring funds requires out-of-band mechanisms to deal with checks returned for insufficient funds, or because they were forged or mis-drawn, but the same is true in the real world.

The second approach to transferring funds is used when a server requires a guarantee that sufficient funds are available. This will often be the case when maintaining quotas. The approach is analogous to that of a certified check. In this approach, the accounting server acts as an authorization server. The client must request an authorization proxy (certified-check) to be used at the end-server. The request would include the name of the end-server, the currency, and the maximum that the payee is willing to pay for the operation. If sufficient funds exist, the accounting server will return the certified check to the client and place a hold on the funds in the client's account. The client will present the authorization proxy to the end-server along with its request.

Once the operation has been performed, the end-server forwards a bill to the accounting server together with a copy of the certified check. The accounting server will then look for the check in its list of outstanding certified checks, and if found, make the transfer.

4 Related Work

In this section I briefly mention other work that has been done in the area of distributed authorization and accounting.

Sun's Yellow Pages allows for the distribution of files such as **/etc/group** which may be used for authorization purposes. In the yellow pages scheme, however, the authorization decision is still made on the local system. With distributed authorization and group services as supported by restricted proxies, the authorization decision can be delegated to a remote server.

There has been much work concerning the forwarding or delegation of authentication in distributed systems. In [3] Karger proposes a server that keeps track of special passwords that are established when a user logs in. These passwords may be passed to other systems which are to act on the user's behalf when an operation involves 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.

The proxy mechanism described in this paper is very similar to the delegation mechanism supported by the Digital Distributed System Security Architecture [1, 2] in which principals generate and sign delegation certificates to allow intermediate systems to act on their behalf. One of the primary

differences is that in the DSSA, restrictions are only supported 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 would then only support access that is specifically authorized for that role. Creating new roles is too cumbersome for generating restricted proxies on the fly or which are to grant access to individual objects. Roles could not be used to implement the authorization server described in Section 2.2.

The mechanism that comes closest to restricted proxies is the cascaded authentication mechanism described by Sollins[8] in which restrictions can be added as credentials are passed from system to system. The differences between Sollins' approach and support for cascaded authorization based on restricted proxies was described in Section 2.4.

Work is underway on the Open Software Foundation's Distributed Computing Environment that makes use of Kerberos [5] to pass authorization information. Although based on a different model, the implementation of their authorization server is similar to that described in Section 2.2. In fact, authorization information is passed using a field that was added to the Kerberos credentials specifically to support the restricted proxies described in this paper.

Surprisingly little attention has been paid to the issue of accounting in distributed systems. A paper on Sentry[7] lays the ground work for an accounting 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[6] 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 also supports multiple currencies.

5 Integration with Existing System

It is straightforward to support restricted proxies on top of existing authentication systems. In this section I show how they may be implemented on top of authentication systems based on both public key and conventional cryptography. I also show how Version 5 of the Kerberos authentication system has been modified to better support restricted proxies.

5.1 Two Ways to Use Restricted Proxies

There are two classes of proxies: bearer proxies and personal proxies. A bearer proxy may be used by anyone, but a personal proxy may only be used by the named principal, or someone with a proxy issued by that principal. When a personal proxy is used, the grantee must authenticate itself

to the end-server. When a bearer proxy is used, the bearer must prove to the end-server that the proxy was legitimately received, and not obtained by eavesdropping on the network.

All proxies have an associated session key. The proxy may be sent across the network in the clear, but the session key must be protected. When a proxy is passed from one principal to another the session key is passed along with it. If the use of the proxy requires the principal to prove that the proxy was received legitimately, it can do so by proving that it possesses the session key associated with the proxy. This is accomplished using the final exchange from the authentication protocol on which the proxy mechanism was built[5].

5.2 Restricted Proxies Using Public Key Cryptography

A proxy implemented on top of a public key authentication system contains the name of the grantor, a session key[6] generated by the grantor, the expiration time for the proxy, and any restrictions on its use (see Section 6). All fields except for the name of the grantor are encrypted in the grantor's private key[7].

The proxy is then passed to the grantee. When presented to the end-server, the end-server decrypts the proxy using the public key of the grantor (obtained from the authentication/name server), verifies that it is authentic, accepts additional authentication from the grantee (either personal authentication or proof that it knows the session key), checks the restrictions, and if all checks out, performs the requested operation.

5.3 Restricted Proxies Using Conventional Cryptography

A proxy implemented on top of an authentication system based on conventional cryptography contains the same information as one based on a public key authentication mechanism. The proxy must be accompanied, however, by credentials authenticating the grantor to the end-server. The proxy itself will be encrypted in the session key from the credentials[8].

5.4 Restricted Proxies in Kerberos

The discussion so far has described restricted proxies independent of the details of any particular authentication mechanism. In this section I describe features that were included in Version

[5]The part of the authentication protocol that supports key distribution is not required since the session key is encrypted within the proxy and is already available to the end-server.

[6]The session key included in the proxy would be one key from a public/private key pair. The session key that is passed to the grantee of the proxy would be the other key from that pair.

[7]Note that if a mixed authentication system is in use where the session key is a secret key from a conventional cryptosystem, then the session key must also be encrypted in the public key of the end-server.

[8]The session key from the credentials was generated by the authentication server. The session key in the proxy is different and was generated by the grantor of the proxy.

5 of The Kerberos Authentication System to provide better support for restricted proxies. Although restricted proxies could have been supported in Version 4 of Kerberos using the method just described, the explicit support for proxies in Version 5 makes their use more transparent to applications which have already been modified to use Kerberos.

Kerberos [5, 9] is an authentication system based on conventional cryptography, designed as part of MIT's Project Athena. 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[4] is the first major revision of the protocol since its original release. Version 5 contains several new features that are important for the support of restricted proxies.

A Version 5 ticket has 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 credentials. The Kerberos protocol does not specify how the sub-fields are to be interpreted except to stress that restrictions must be additive. Each subfield can only place additional restrictions on the use of credentials, never remove restrictions or grant additional privileges.

When credentials are requested, the requesting principal can specify that restrictions be placed on their use. If credentials are issued based on existing credentials, restrictions may be added, but not removed. The remaining details of the proxy mechanism (as described earlier) are subsumed by the authentication mechanism itself.

5.5 Discussion

There are several advantages to supporting proxies within the authentication mechanism instead of on top of it. As already stated, transparency is one of the reasons. The second advantage 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 implementing proxies using conventional cryptography is that each proxy can be used at only a particular end-server[9]. This disadvantage is offset somewhat by the difficulty of

[9]This disadvantage also applies to the mixture of conventional and public key systems where the proxy's session key is from a conventional cryptosystem.

revoking a proxy that can be used anywhere.

Implementing proxies within Kerberos itself allows one 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. This eliminates the disadvantage of basing the proxy mechanism on conventional cryptography.

6 The Restrictions

Regardless of the method used to implement restricted proxies, the interpretation of the authorization field must be specified. The authorization field should be interpreted as a collection of typed subfields. Each type corresponds to a different restriction. This section describes the restrictions that should be supported.

6.1 Grantee

This restriction includes the identity of the principal which is authorized to use a proxy. In order to use a proxy with this field specified, the authorized principal must present its own authentication credentials[10] 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. In the case of a bearer proxy, the bearer must present a corresponding authenticator to prove that it also possesses the session key associated with the proxy, thus preventing an attacker from using a proxy obtained by eavesdropping on the network.

6.2 Issued-to

The **issued-to** restriction is not really a restriction. Instead, it is informational only. It would be included in a bearer proxy to identify the principal to whom the proxy was issued. It could be used by the end-server to maintain an audit trail. A proxy with more than one **issued-to** field should be considered invalid, and a principal issuing a bearer proxy should strongly consider including this field.

It is expected that the most common use of this field might be in bearer proxies issued by an authorization server. Such a proxy could be used without requiring a grantee to further authenticate itself to the end-server, but the issued-to field would still allow the end-server to know on whose behalf it is really performing the operation.

[10] Or an additional proxy granted by the authorized principal.

6.3 Issued-for

The **issued-for** restriction specifies the identity of the end-server. It is only necessary if the method used to implement proxies supports proxies that may be used for more than one end-server.

6.4 Require-Authenticator(this,last)

When the grantee subfield is specified, the authenticator associated with the proxy is usually not required. The authenticator corresponding to the personal credentials of the grantee must still be presented and is sufficient. There are cases, however, when the authenticator corresponding to such a proxy should be required. A prime example is in the case of the standing proxy described at the end of section 2.4. Although the subordinate server possesses a standing proxy it should still have to prove that the intermediate server passed it the credentials to which it is attaching the proxy. Without such a requirement, the subordinate server could use a ticket obtained through other means (such as by eavesdropping on the network).

Adding the **require-authenticator** restriction to a proxy specifies that the bearer must include an authenticator corresponding to the proxy even if the grantee restriction has been specified. An argument to this restriction indicates that the authenticator restriction corresponds, not to this proxy, but to the previous proxy in a chain of cascaded proxies. It is this form that would be needed in the standing proxy example.

6.5 Accept-Once

The **accept-once** restriction tells an end-server that it is only to accept a proxy one time. The end-server would have to maintain a list of all one-time proxies and check the list each time one is received. Once used, the proxy would have to remain in the end-servers list until the expiration time of the proxy.

The **accept-once** restriction can take an identifier as an argument. If present, only the identifier and the name of the grantor need be stored in the end-server's list of used proxies. Any other proxy bearing the same identifier, and received by the end-server within the expiration time of the first proxy would be rejected. A real life example of such an identifier is a check number.

6.6 For-use-by-Group

The **for-use-by-group** restriction specifies that the proxy may only be used by a member of the named group. To use such a proxy, the bearer would have to present the proxy along with an additional proxy from the appropriate group server.

6.7 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.

6.8 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. It is important to keep in mind that this restriction limits 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.

6.9 Accompanying

This restriction requires that the proxy can only be used when accompanying another proxy. The restriction can identify the proxy which must be accompanied by including the principal identifier of the grantor, and the issue time of the credentials. A standing proxy, would require an accompanying proxy, but the accompanying proxy would not be identified. A standing proxy will usually be issued with the **accompanying** and the **require-authenticator**(last) restrictions specified.

6.10 Authorized

The **authorized** restriction specifies a complete list of those objects which may be accessed using the rights granted by a proxy. This restriction will usually be specified when individuals grant a proxy or generate a capability. It will also be specified in a proxy returned by an authorization server. There should be no constraints on the form of the names of the object[11]. They are to be interpreted by the end-server.

6.11 Rights

The **rights** restriction is identical to the **authorized** restriction except that each object is followed by a list of access rights. As was the case with name of the object, there are no constraints on the form that the access rights can take. They are to be interpreted by the end-server.

[11] Though certainly the grantor and the end-server must agree on the form.

6.12 Write-above

If implementing multi-level security the **write-above** restriction restricts the use of the proxy for writing objects to those objects above (inclusive) the specified security level. This restriction is in addition to all other restrictions on access to data.

6.13 Read-below

If implementing multi-level security the **read-below** restriction restricts the use of the proxy for reading objects to those objects below (inclusive) the specified security level. This restriction is in addition to all other restrictions on access to data.

6.14 Limit-restriction

Restrictions that are defined only for particular end-servers might be needed. If a proxy can be used for a server that does not understand a restriction, the restriction or restrictions must be associated with the name of the server to which the restrictions 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.

6.15 The Propagation of Restrictions

Certain servers (authentication, authorization, and group servers for example) 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.

7 Discussion and Conclusions

The problems of authentication, authorization, and accounting are more similar than most people realize. 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.

This paper has shown how restricted proxies can be supported in existing authentication systems, and it has shown how they can be 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 Ed Lazowska, David Keppel, and Steve Bellovin who commented on earlier drafts of this report.

References

[1] 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, 1989.

[2] 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.

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

[4] John T. Kohl and B. Clifford Neuman. The Kerberos network authentication service. DARPA Internet Draft RFC, December 1990.

[5] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer. Section E.2.1: Kerberos authentication and authorization system. Project Athena Technical Plan, MIT Project Athena, Cambridge, Massachusetts, December 1987.

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

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

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

[9] 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. Dallas, Texas.