Access Control in an Open Distributed Environment

R.J.Hayton

APM Ltd
Poseidon House
Castle Park
Cambridge
United Kingdom

J.M.Bacon, K.Moody

University of Cambridge
Computer Laboratory
Pembroke Street
Cambridge
United Kingdom

Abstract

We describe an architecture for secure, independent, interworking services (Oasis). Each service is made responsible for the classification of its clients into named roles, using a formal logic to specify precise conditions for entering each role. A client becomes authenticated by presenting credentials to a service that enable the service to prove that the client conforms to its policy for entry to a particular role. During authentication a data structure is created that embodies the proof.

An authenticated client is issued a role membership certificate (RMC) for its subsequent use with that service. An RMC is an encryption-protected capability which includes the role name, the identity of the principal to which it was issued and a reference to the issuing service. A proof rule of one service may refer to an authenticated user of another; that is, an RMC issued by one service may be required as a credential during authentication by another. A dynamic proof tree may thus be built which exhibits amongst other things the trust relationships between the services which the client has entered.

The first part of the paper shows how a service may define a set of proof rules (Horn clauses) that specify who may use it and in what way. Delegation of rights may be expressed naturally within these rules.

The second part of the paper presents the design details of the system. Associated with each RMC issued by a service, the service keeps a credential record (CR). The CR indicates the predicates against which the RMC was issued and lists all other services which have issued RMCs to this principal based on this CR. If one of these predicates becomes false, the local RMC is immediately invalidated. Event technology is used to achieve rapid revocation of the dependent RMCs issued by other services; any portion of a proof tree which is based on this predicate collapses.

The system is inherently decentralised and has a tuneable reaction to network or server failure which allows services to make appropriate decisions when authorization or revocation information is unavailable.

A prototype system has been implemented and tested.

1 Introduction

The access matrix

Access control is often presented as the enforcement of policy described by an *access matrix*. The columns of the matrix represent the operations that may be performed on the protected object or service, and each row represents a client. For each client there is therefore a complete description of which operations that client may undertake, and likewise, for each operation we may enumerate all clients who have access. The access matrix is conceptually simple and the majority of access control models are based on it.

In order for the access matrix to represent a particular policy, we must enumerate who the potential clients are. In an open environment any process may attempt to perform any exported method on any object.

Our approach

We argue that it is insufficient simply to enumerate all possible clients of a given service, but that those clients should be classified by that service according to service specific semantics.

For example a login service may use categories *logged-in user*, *logged-in professor*, *logged-in student* to distinguish between classes of client. Within these categories it may distinguish individuals by using identifiers, for example **User**(`Jean`) or **Student**(`Fred`, `Math`). Equally, a conference service may use the categories *Chair* and *Member*, and a bank account service may use the categories *customer* and *credit card authority*. If services are responsible for this naming of the roles in which their clients may operate, then revocation and delegation can be formulated in terms of the granting and revoking of these names. This allows the authors of services to express application specific semantics clearly.

We note that the way in which clients are classified for use in the access control statements of one service will not be suitable for use in all services. It is therefore essential to allow *each* service to perform naming and authentication of its clients in order to authorize them as its named users. It should be possible to capture the trust relationships between services. For example a conference service should be able to express "if the login service guarantees you are logged in as user *jb* or user *km* then you may become a named user *Chair* of this (conference) service". That is, a principal may be issued with a name by one service on condition that it has already been issued with some specified name of another service. A proof tree may therefore be built corresponding to the trust relationships between the services which it spans. Our implementation mechanisms maintain such proof trees dynamically. A principal which has satisfied the conditions for possession of a name is issued with a *Role Membership Certificate* (RMC), and the issuing service maintains an associated *credential record* (CR) dependent on those conditions.

With this new approach to access control, issues such as delegation and revocation have clear semantics. For example the policy of the conference service may be that only someone who has the name *Chair* (having therefore satisfied the authentication check) has the right to speak but may delegate this right to someone with the name *member*. Policy may therefore be expressed to a higher degree of precision than is possible in pure access matrix based systems.

If a condition for holding a role name becomes false any service that depends on this condition, directly or indirectly, should be informed immediately. We use event technology to signal such a predicate violation, thus collapsing any portion of the proof tree which is no longer valid. This mechanism is used to implement rapid and selective revocation of privileges at all the affected services in the distributed system.

Distributed systems are subject to independent failure of component sites. We provide a heartbeat protocol, which may be tuned to arbitrary time granularity. We can thus distinguish between the absence of a predicate violation event and failure of the sending site. A highly sensitive service may suspend the activity of a name that depends on the continuing validity of a credential held by a failed system, whereas other services may choose to allow potentially affected names to continue to operate. Again, we provide a general, tuneable mechanism on which a variety of policies can be based.

The approach is naturally distributed, with each object or service responsible for its own authentication and authorization. This allows decentralised administration, a major goal of open systems research. A simple role definition language (RDL) is provided which allows each service to express the policies that govern the conditions for its use.

Section 2 describes related work. We believe that no other system has achieved the fine-grained control, generality and openness of our approach. Part 1 of the paper, sections 3 through 7, is about specifying access control policy. Section 3 discusses the nature of the clients of a service. Section 4 shows how services may define role names independently. Section 5 defines a logic and its associated language in which access control policy may be expressed; a *role definition language* (RDL). Section 6 shows how access rights may be withdrawn by selective revocation. Section 7 presents access control for a filing system as a short case study.

Part 2 of the paper discusses the implementation of this design. Section 8 describes the structure of a role membership certificate. Section 9 shows how event driven software technology can be exploited to achieve immediate and selective revocation of rights. Section 10 describes credential records, which form the heart of the implementation strategy. Section 11 discusses issues specific to the distribution of the mechanisms. Section 12 discusses how failures are handled within the model. Each service may decide its own policy when a source of credentials used during authentication has failed. Section 13 summarises and concludes the paper.

2 Related Work

The seminal work by Lampson [1] established the ground rules for access control policy specification and implementation mechanism. The formulation of access control in terms of client naming has its roots in existing role based access control architectures, such as are described in [2]. However these models use the term *role* as a pseudonym for user, which is less general than the approach presented here. Other attempts at the formal specification of access control policy are Taos [3] and [4]. We believe that RDL statements are more natural and more comprehensible than these approaches.

Existing models tend to rely on a single, per organisation policy to define who undertakes each role, and what the relationships between roles are [5]. We see roles as more widely applicable and more flexible, capable of being defined independently by services inhabiting multiple administrative domains in an open world. For this style of use we need a model that supports application specific roles, and that captures the relationship between the roles defined by and used within different contexts. In an open environment it is important that policy statements capture the dynamic nature of access control. If this is not the case, and policies must be manually updated to reflect changes in access rights, then it is difficult to reason about how access right might propagate through a system, or to prevent error.

Requirements for security in open systems are set out in [6]. The paper describes a scalable and open mechanism for enforcing security, but issues of policy expression are not discussed. The mechanism is based on capabilities and has the traditional features of flexible delegation, but poor revocation. The Oasis design includes a logic for policy expression and the implementation provides efficient and selective revocation.

Part 1: Specification of access control policy

3 What makes a client?

The terms *user* and *client* are often used synonymously. However, requests to a service do not come directly from users - they come from *processes*. Although it is often a simplification to assume that the request was initiated by a user, this is not always helpful - especially if the process making the request is executing a malicious program masquerading as a benign one.

There are several classes of attribute a process might possess that are generally useful for classifying it as a client of a particular service. Firstly, the process may have already been classified by the same, or some other, service. For example, if a client has been classified by a login service as representing a particular user, then that information is likely to be of use to other services.

The second attribute is evidence of delegation. For example a process representing a user might allow a process running a Java applet to read a file that it would not otherwise have access to. Whether this delegation is honoured will depend on the protection policy governing access to the filing system. In terms of Oasis, rights themselves are not delegated. Delegation is used to supply potential clients with additional attributes that allow them to be accepted as clients by a particular service. As will be seen, delegation of rights is simply a special case of this.

Finally the process may have delegated rights revoked. For example, we may revoke the rights given to the Java applet once we believe that it has fulfilled the intended task.

In addition to these attributes, there are many others that may be applicable to particular services: a file service may give special privileges to the *owner* of a file; a bank service may need to be invoked by running a trusted program; a news feed may allow access by students only at weekends. A flexible access control model must be able to deal with each of these circumstances without inflicting an unnecessary cost on all services.

If the clients of a service are carefully classified, the access matrix itself is often extremely simple. Only interactions between clients and objects need be considered. Interactions between clients such as delegation and revocation are subsumed by client naming.

4 Allocating Names

Policy specification in Oasis is based on sets of rules which indicate the conditions under which a client may obtain a name or *role*. As a client runs, it will gather credentials issued by different services. It may for example obtain a certificate to show that it has successfully supplied a password, or a certificate that it is executing a particular program text. The client may then approach an Oasis service and ask to be granted membership of a role. The service will consult its policy and, if appropriate, will grant the client a role membership certificate (RMC) indicating that it is now a member of the requested role. This RMC may then be used directly by the client when performing operations, or alternatively it may be presented to the same or another Oasis server to be used as a credential when entering a new role.

In existing access control architectures, the issuing and revoking of rights or names is considered as a core facility. The services responsible for such actions are generally highly secure and run with special privilege. Oasis is an open, distributed architecture, and the designers were unwilling to make such assumptions. For this reason, Oasis servers are designed to be essentially independent, and there need be no global consensus about the trustworthiness of clients, or certificates issued by other services. On the other hand, a particular Oasis service may place limited trust in other services. For example, a service allocating organisational roles, such as Manager and Service Engineer, is likely to rely on certificates issued by a Login service. This provides a useful level of abstraction. The organisational role service is interested in the validity of a Login certificate, and relies on the correct issue and revocation of such RMCs; it is not concerned with the details of how or why such certificates are issued. This decouples the specification and administration of the conditions for Login, from the specification and administration of the conditions for organisational role membership.

In general, there may be many services issuing Oasis certificates for different, unrelated, purposes. For example, each instance of a conference application may issue certificates indicating who may act as the Chair of the meeting, and who the members are. It is an important feature of the Oasis architecture that there need be no trust between different instances of this application, and that the conference service need not be trusted by other unrelated services and applications.

5 Policy Expression in Oasis

Each Oasis service is at liberty to use its own means of policy expression and method for realising that policy; to conform to Oasis a service need only issue certificates and maintain their validity. However, there are many reasons why it is advantageous to use a common means to express policy. First, it is important that policies may be easily understood by administrators and users; if this is not the case, then errors are inevitable. Second, it aids reasoning about the interaction of policies between different services; this reasoning may be performed by hand, or with the aid of automated tools such as theorem provers.

For these reasons a *role definition language* (RDL) was developed that allows the flexible and unambiguous specification of access control policy. An RDL statement may be considered as an axiom in a proof system, aiding both human and automated reasoning.

Each RDL statement defines a set of conditions which suffice for a client to enter a role. In addition, by annotating the clauses of the statement, the circumstances in which this role should be revoked may also be specified. The grammar for RDL is given in figure 1, and there is an example below. The conditions for entry to a role are described in terms of the credentials that establish a client's suitability, together with constraints on the parameters of those credentials.

For example, an examination service may wish to indicate that a client may enter the role ChiefExaminer if they provide credentials proving that they are logged in as user `km` on a trusted machine. In Oasis terms this corresponds to the client possessing a certificate of the form **LoggedOn**(`km`, s) issued by the Login service, where s is a trusted server. This may be written in RDL as

$$\begin{aligned}\textbf{ChiefExaminer}() \leftarrow{} & \textbf{Login.LoggedOn}(\texttt{km}, s)\\ & : s \text{ in } \texttt{TrustedServers}\end{aligned}$$

Role Declaration

def **Rolename**(*arg*,...) *arg* : type [*arg* : type ...]

Type import

import **ServerName**.`typename`

Role Entry Statements

Role Name([**arg**, ...])←**Role Reference** [∧ **Role Reference** ...] [: **Constraint**]

Role Name([**arg**, ...])←**Role Reference** [∧ **Role Reference** ...] ◁ **Delegator's Role Reference** [: **Constraint**]

Figure 1: Grammar for RDL

Role declaration and Type import statements are used to provide naming information to the RDL parser. As RDL has a comprehensive type inference scheme, these are rarely required. Role entry statements take two forms, 'normal' role entry based on the presentation of role membership certificates, and 'delegation' for specifying entry through delegation or election.

This is equivalent to the axiom

$$\frac{\begin{array}{c} x \text{ requests } \textbf{ChiefExaminer} \text{ based on certificate } C \\ C \text{ was issued by the Login service and is still valid} \\ C \text{ is of the form ``} x \text{ has } \textbf{LoggedOn}(\texttt{km}, s)\text{''} \\ s \text{ in } \texttt{TrustedServers} \end{array}}{\begin{array}{c} x \text{ may be issued with a certificate of the form} \\ \text{``} x \text{ has } \textbf{ChiefExaminer}\text{''} \end{array}}$$

As well as role membership certificates, an RDL statement may also require a client to present evidence of delegation (or election).

For example, a RDL statement indicating that "the Chief Examiner may elect any logged on user who belongs to the group `Staff` to be an Examiner, specifying the examination subject e" may be expressed as

$$\begin{array}{l} \textbf{Examiner}(e) \leftarrow \textbf{Login.LoggedOn}(p,s) \\ \qquad\qquad \lhd\, \textbf{ChiefExaminer} : p \text{ in } \texttt{Staff} \end{array}$$

The symbol "◁" is read "delegated by".

Delegation in Oasis is considerably more general than in other models where only rights, not roles, may be delegated. A client may delegate a role that they do not themselves possess, if this agrees with policy. Such a delegation might more accurately be termed 'election'. Like role membership, proof of delegation is achieved by presenting a certificate. The above rule is equivalent to the axiom

$$\frac{\begin{array}{c} y \text{ requests } \textbf{Examiner}(e) \text{ based on } L, D, C, \{E\} \\ L \text{ was issued by the Login service and is still valid} \\ L \text{ is of the form ``} y \text{ has } \textbf{LoggedOn}(p,s)\text{''} \\ D \text{ was issued by this service and is still valid} \\ D \text{ is of the form ``} z \text{ delegates } \textbf{Examiner}(e) \text{ to } P\text{''} \\ \text{``} y \text{ has } P\text{'' is provable from } \{L, E\} \\ C \text{ was issued by this service and is still valid} \\ C \text{ is of the form ``} z \text{ has } \textbf{ChiefExaminer}()\text{''} \end{array}}{\begin{array}{c} y \text{ may be issued with a certificate of the form} \\ \text{``} y \text{ has } \textbf{Examiner}(e)\text{''} \end{array}}$$

Note the use of the constraint P on the delegation in the above axiom. This is used to allow the delegator to specify a constraint on the client using the certificate, for example the Chief Examiner would typically constrain the delegation to apply to a particular user by insisting that that client supplied an additional Login certificate. This is more robust than relying on the delegation certificate being passed to the correct client, and in addition allows a delegation certificate to be used by a number of processes that represent the same client. This is particularly useful if a delegation certificate is to be used over a long period of time, for example if a Lecturer delegates to a colleague whilst on sabbatical leave.

6 Revocation

The discussion of RDL so far has concentrated on specifying conditions for entering a role. It is just as important to specify the circumstances in which a role membership should be revoked. It is not sensible to revoke a membership if the client can immediately re-enter the role. Revocation should therefore take place only when the client is no longer eligible to enter the role. It could be argued that these are precisely the conditions for revocation, but we believe that a more flexible approach is better. In the above examples, role entry is validated by checking that a number of *entry conditions* hold. The conditions for revocation are specified by indicating, for each condition, whether it must continue to hold throughout role membership or may cease to hold without causing revocation. Explicit revocation (where the delegator wishes to revoke) is modelled as the revocation of the delegation certificate and this, like any other clause, may or may not cause revocation according to policy. Extending the previous example to show how a student may become a candidate in an examination:

$$\mathbf{Candidate}(p,e) \leftarrow \mathbf{Login.LoggedOn}(p,s) \\ \triangleleft\, \mathbf{Examiner}(e) : p \text{ in } \mathtt{Students}$$

There are four entry conditions that may change over time. The RDL must therefore be annotated to indicate those whose failure should result in revocation of the Candidate certificate. The four conditions are

- that the client is logged on as user p on host s
- that the delegator holds the Examiner role for subject e
- that the delegator has not revoked the delegation
- that p is a member of the group Students

If revocation should take place if the delegator revokes, if the client logs off, or if the student is suspended, then the annotated rule is

$$\mathbf{Candidate}(p,e) \leftarrow \mathbf{Login.LoggedOn}(p,s)^* \\ \triangleleft^*\, \mathbf{Examiner}(e) : (p \text{ in } \mathtt{Students})^*$$

This technique is both straightforward and flexible. In the following section we consider how RDL may be used to specify a more traditional form of delegation, and in the second part of the paper we show how such a statement is translated to a direct and efficient implementation.

7 Case Study: Policy expression for a Filing System

The previous section explained how RDL may be used to specify how clients may obtain roles. In this section we show that certificates subsume traditional filing system capabilities, so that RDL may be used for the specification and implementation of policy within a filing system.

A filing system is unusual in that it maintains a large number of policies, typically one set of policies per file. In a traditional system, such policy statements take the form of mappings between users and rights, and are called *access control lists.* The rest of the policy, relating to the delegation and revocation of rights, is usually implicit in the implementation and is not spelt out in policy statements. This is clearly a disadvantage when reasoning about policy interactions.

Under Oasis, each ACL may be replaced with a set of RDL statements which indicate the classes of client to be granted access, and map these to a role representing the capability to access a file. As we typically require many levels of access, this role may be parameterised with a set variable indicating the operations that are permitted. For example the role

$$\mathbf{UseFile}(\{\mathtt{read}, \mathtt{write}, \mathtt{delegate}, ...\})$$

might be appropriate. In order to use a file, a client first requests a UseFile role membership certificate, supplying as argument their Login certificate or other credentials, and in return is granted a certificate to be used as a capability when performing operations on that file. A RMC certificate may only be used by the process to which it is issued, so there is no need to parameterise the **UseFile** certificate to indicate the user for whom it is intended. Equally, RMCs are specific to a service context. In this example, the service context is the file to which the RMC pertains. This scheme is more flexible than a traditional ACL as roles may be issued against credentials other than user identity or group membership. For example the rule

$$\mathbf{UseFile}(\{\mathtt{read}\}) \triangleleft\, \mathbf{Meeting.member}(x)$$

attached to a file would allow members of the meeting in the earlier example to enter a role issued by the (mutually untrusting) file service in order to read the meeting's minutes. Although more flexible, RDL statements are also more complex than traditional ACL statements, and in practice a shorthand may be provided that is automatically expanded to RDL when required.

As file access policy is now described in terms of RDL, we can specify a policy on the delegation and revocation of access rights. As an example, imagine that we wish to allow recursive delegation between logged in users, but that we wish to restrict delegation to other clients to be read only. We may then use the rules

$$\begin{array}{lll} \mathbf{UseFile}(x) & \leftarrow & \mathbf{Login.LoggedOn}(u) \\ & & \triangleleft\, \mathbf{UseFile}(y) : x \subseteq y \\ \mathbf{UseFile}(\{\mathtt{read}\}) & \leftarrow & \triangleleft\, \mathbf{UseFile}(y) : \mathtt{delegate} \in y \end{array}$$

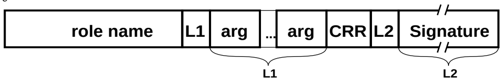
A more detailed discussion of this case study may be found in [7, Chapter 5].

Part 2: System design and implementation

In the previous sections we have considered the flexible specification of access control policy in terms of client naming. In the rest of the paper we consider how such a scheme may be implemented.

8 Format of a Role Membership Certificate

Each RMC consists of a role name, a set of parameters to that role, a 64 bit record used for revocation purposes called a *credential record reference* (CRR), and an optional signature to secure it cryptographically.



Each RMC may be used by only one process, and protection against stolen certificates is provided by including the process identity when calculating the signature, as in [8]. The CRR is associated with a credential record (CR) held by the issuing service (see below); each RMC and each delegation certificate have a single associated CR.

The format of the certificates is quite general since they are designed for use by many interworking services that require different numbers and types of parameter. In particular, each service is at liberty to use a different mechanism for signing their certificates. This allows the designers of each service to make appropriate tradeoffs in terms of signature length, computational cost and security.

A client must present an appropriate RMC, issued by that service, with each request to use a service. The service must validate the signature, to ensure that the certificate has not been tampered with, and to check the client process's identity. It will then use the CRR to locate the CR associated with the RMC. The CR indicates whether the certificate has been revoked. The issuing service will then inform the client application of its decision. As we have seen in Part 1, a service may need to verify a number of RMCs issued by this and other services before issuing an RMC to a client. Each RMC presented must be validated by the service which issued it.

As the integrity check against modification and the check for revocation use different techniques, the result of the cryptographic check may be cached by the service to avoid recomputation if the same RMC is used repeatedly. Revocation must be checked via the CR on every use. An efficient mechanism for maintaining revocation state in the CRs is described in the next section.

9 Event Driven Evaluation

In Part 1 we described how a client may collect certificates indicating that it is a member of a number of roles, and how these certificates may be presented to a service in order for the client to enter further roles. This process was likened to establishing a dynamic proof of role membership as illustrated in figure 2.

If certificates were never revoked, implementation would be straightforward; each service would validate the RMCs presented and issue further certificates if appropriate. One approach to revocation management, used in Taos, is to discard certificates periodically, thus forcing clients to acquire new certificates and re-enter roles. This can lead to a costly implementation, especially if there are long chains of membership. In addition, the lifetime of a certificate is critical, as revocation cannot take effect until that lifetime expires. The fact that revocation is not immediate is a security loophole.

Oasis takes a different approach. The proofs established by application of RDL rules are assumed to hold indefinitely until some *event* occurs that

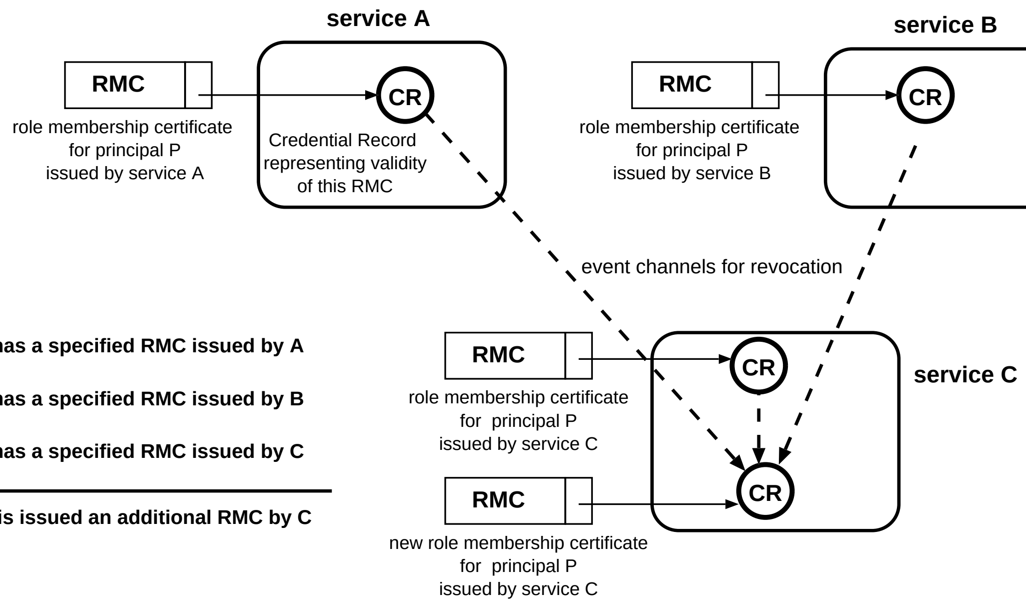


Figure 2: Entering a Role

leads directly to a change. Data structures representing the proofs are established in such a way that when an event occurs, precisely the dependent portion of the proof tree will collapse, without affecting the validity of other RMCs.

To achieve this we use a single CR to represent a server's current knowledge about each individual fact. A CR is a small record stored with the issuing Oasis service to record that a fact is *true*, *false* or is currently *unknown* by that service. Events are precisely those occurrences that lead to a change in the state of some CR. We arrange that each revokable delegation, group membership or other credential that may change is represented by a CR. When a credential changes, its CR is modified and this information is propagated, so that all dependent CRs issued by the service are also modified. Each certificate is dependent on the state of exactly one CR, which allows the service to validate certificates rapidly. If some event occurs requiring that the certificate should be revoked, then the CR will be set to *false*, and validation of the certificate will fail.

In a distributed environment, certificates issued by one service frequently depend on the validity of certificates issued by another. During authentication, the service carrying out the checks will ask the service issuing any imported certificate to verify the certificate, and to *notify* it if the state changes subsequently. This can considerably reduce the amount of network traffic, and also the access latency of authorization checks. This is considered in detail in the following sections.

10 Credential Records

The CRR is a reference to a unique CR within the issuing service. When an RDL statement is invoked in order to enter a role, then a graph of CRs can be constructed to represent the membership rules involved. For example, in figure 3 there are three revokable credentials; the login certificate, a current group membership and an indication of the Examiner's willingness to delegate. The certificates involved all contain references to the related CRs, and the dependencies can easily be included in the proof graph.

10.1 Format of a Credential Record

The format of a credential record is shown in figure 4. CRs are stored in a large table within a server. Whenever a table entry is reused, the

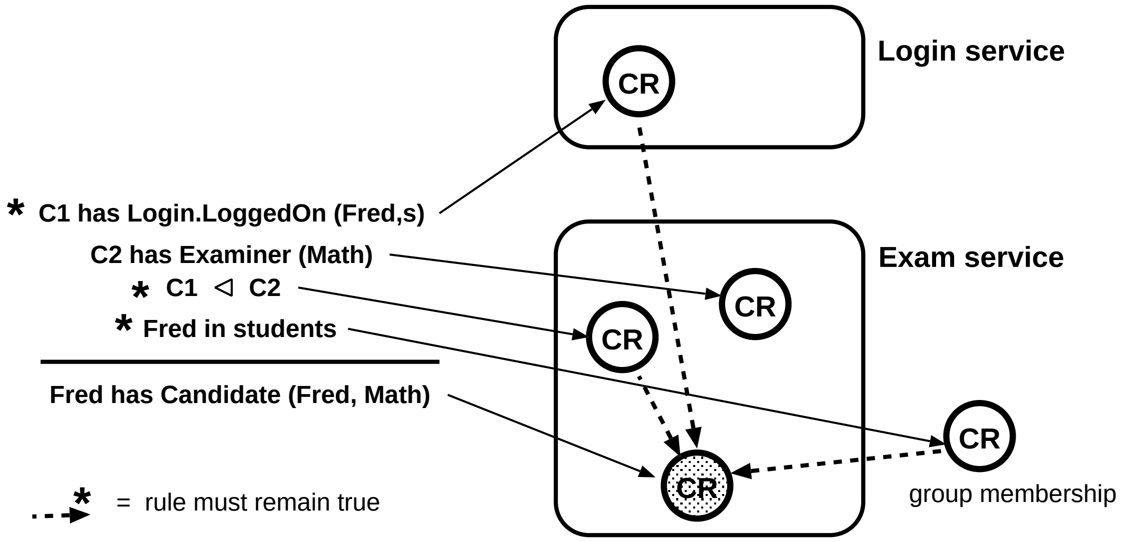
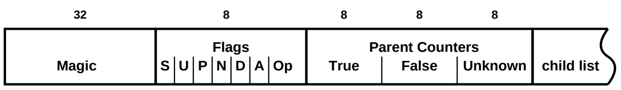


Figure 3: Entering a Delegated Role

A client entering the role Candidate must supply a Login Certificate and a Delegation Certificate. From the references embedded in these two certificates, and the one found by group membership lookup, the illustrated graph can be constructed. The shaded record represents the logical conjunction of the membership rules; a reference to this credential record is included in the membership certificate. The arcs represent the membership rules which must remain true.



Flag	Description
State	Current truth value, True or False
Unknown	Value is currently unknown due to network failure
Permanent	The state will never change
Notify	Another service is using this credential
Direct Use	A certificate has been issued that embeds this credential
Auto Revoke	This credential should be revoked if a parent exits
Op	Binary operation performed on parent values

Figure 4: Format of a Credential Record

`Magic` field is incremented, thus ensuring that (`table index`, `Magic`) forms an identifier which is unique over the life of the service. This tuple is used to form a 64 bit identifier which becomes the CRR. This method can also be used for CRs stored in persistent store, and is described in more detail in [9, 6.4].

Each CR stores a list of its children, so that when its state changes the information can be propagated to its dependents. Their values will be changed by this means and the process will recurse, if necessary. Rather than storing backwards pointers from child records to parents, a more efficient scheme is used, and counters are kept of the number of parents that are *true*, *false* or *unknown* due to network failures. This information is all that is required to set the state of a record.

A *permanent* flag within each record indicates if state changes are possible. Whenever a state change is not possible, for example after revocation, the record itself is redundant and can be garbage collected. A record may also be deleted if it represents a fact that is *uninteresting*, i.e. one that has not been used for issuing certificates directly, and that no longer has any child records.

Garbage collection takes place by unlinking parent → child links whenever the value of a parent is made permanent. The record cannot be deleted immediately because this might leave dangling references from its parents. A periodic sweep algorithm unlinks these references, and deletes *uninteresting* and *permanent* records.

10.2 Issuing Delegation Certificates

When a client wishes to delegate membership of a role to another client, it applies to the issuing service for a *delegation certificate*, and a CR is established to record the validity of the delegation. If delegation is to be a membership rule then as a side effect a *revocation certificate* is returned to the delegator; this contains a reference to the CR for the delegation certificate. The delegation certificate is then passed to the second client, who accepts delegation by using the certificate as a credential when entering the named role. Both parties must agree to the delegation, and the delegator can be assured of the ability to revoke, regardless of network failures.

Since clients are named by process identifiers, the candidate client cannot be named directly, but must be identified by specifying constraints on one or more roles that they possess. In our example the Chief Examiner may choose to delegate the role **Examiner**(`compsci`) to a client who possesses **Login.LoggedOn**(`jb`,s). In order to utilize the delegation certificate a candidate client must have both **Login.LoggedOn**(`jb`,s) and the roles required by the RDL statement, ie **Login.LoggedOn**(x,y) where x is in `Staff`. An additional advantage in specifying delegates by the roles that they hold is that delegation may persist for longer than the lifetime of a low-level identifier. It may be appropriate to use one digital signature function for short-lived certificates such as RMCs, and another more secure function for delegation certificates, which are prone to different forms of cryptographic attack. Although long term delegation is useful, it is not always appropriate: for example, a *print* service should only retain access to a file for as long as it takes to print it. A delegator may specify a time limit on the life of the delegation, after which automatic revocation will occur. Revocation may also be specified to take place when the delegator exits the delegating role. When revocation occurs the delegation CR is invalidated, as is the CR relating to any issued certificate. These invalidations are *permanent*, and the CRs may therefore be deleted as described above.

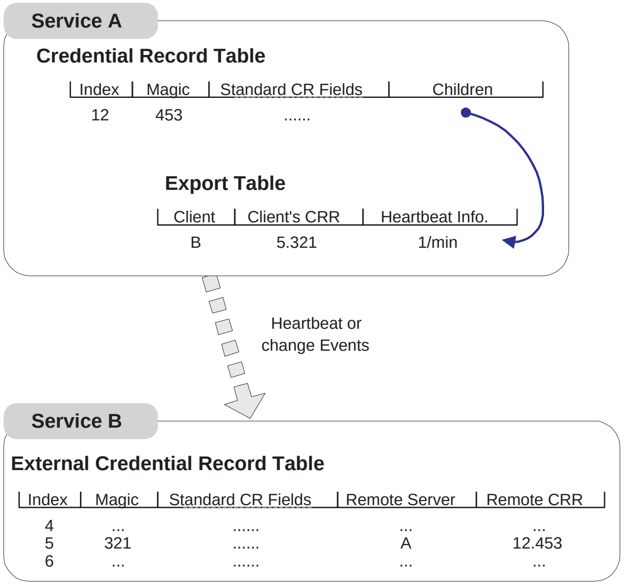


Figure 5: External Credential Records

11 Distributing the Credential Graph

In a distributed environment, certificates issued at one server may be used as credentials at another. Consequently, a CR in one server may be the parent of a record in another. This raises issues of naming, independent failure modes and robustness. In order to decouple the name space and failure modes of two services, *external credential records* are used to represent remote facts and *event notification* is used to communicate state changes between servers.

11.1 External Credential Records

If a server requires a reference to a CR on another service, it creates a local surrogate record called an *external credential record* (ECR). This record contains information about the identity, location and state of the record being represented, together with the standard attributes of a CR, including an identifier within the local name space. The state of the record is maintained by event notification, as described below, and in all other respects the record is treated as a local CR. When the local garbage collector decides that the record is no longer required, the external server is informed so that it can delete corresponding state, including the requirement to notify changes.

11.2 Event Notification

Asynchronous event notification is an important feature of distributed systems, and the RPC mechanism used in the current implementation of Oasis has been extended to add event management functions [10]. Oasis makes use of these functions by defining the event type **Modified**(*CRR*, *newstate*) in the interface definition file of an Oasis server. A server may then register interest in the state of a particular CR, and will be informed if its state changes by being sent an event with *CRR* and *newstate* set to appropriate values. In this way revocation taking place in one server may affect certificates issued by another. The effect of ECRs and event notification is illustrated by figure 5.

12 The Effect of Failures

Event notification between servers may be delayed indefinitely by network congestion or failure. Additionally, either of the parties may fail and restart independently. These situations must be taken into account in the design of any distributed system involving events. The approaches described here are implemented as part of a generic event library, and are equally suitable for any event based application [7].

Consider two parties `A` and `B`, where `A` wishes to send `B` a stream of messages. We assume that the message transport system delivers messages in order, point to point. If every message `A` sends contains a sequence number, then `B` will be able to detect if any *previous* message has been lost. If, in addition, `A` ensures that a message is sent at least once every t seconds, then `B` will know within time t whether a message has been lost or delayed.

This is the basic requirement for event handshaking. In addition, `B` must periodically inform `A` that events have been received, so that `A` may delete any associated state.

This protocol is called a *heartbeat protocol*, and a form of it is used in the event system implemented. The server responsible for signalling events is used as the initiator of the protocol and ensures that a heartbeat event is sent every t seconds. Individual events (and heartbeats) are not acknowledged for reasons of efficiency, but every i heartbeats the client replies, so that the server can detect failures and resend event instances if required.

This leads to a system with the following characteristics:

- A client can be certain of receiving an event within time t of its generation, or of detecting that notification may have failed or been delayed.
- A server can detect that a client is not responding, and after a period, can assume that it is no longer running.
- A client who processes and forwards events can treat heartbeats in a similar manner. This feature allows a service to provide guarantees about 'indirect' events from other services.

In Oasis, a missed heartbeat leads to ECRs being marked as *unknown*. This state propagates to child records, and possibly to other servers. When connection is re-established the state of each record is read and the current state is propagated to its children. This reduces the effect of minor network errors. The period of the heartbeat and the frequency of response can be set on a per-service basis, allowing each service to choose its own trade-off between failure tolerance and security.

13 Summary and Conclusions

We have described an approach to access control in which each component of an open system (a service, say) may *name* its clients according to its internal semantics. The service expresses its policy for the conditions under which a client may acquire a name in a role definition language (RDL). The conditions form a set of Horn clauses (proof rules). A rule of one service may require possession of a name within some other service as a condition. Possession of a name within a service may therefore depend on the existence of a dynamic proof tree spanning several other services.

Such a proof system is ideal for the precise expression of policy. Its implementation is efficient in that acquiring a name within a service can exploit an existing proof tree rather than building a new one; that is, recursive checking of every prior condition in the tree is not needed. The dynamic proof tree should collapse if any of its supporting predicates becomes false. We achieve this by using event driven software technology to signal a false predicate event immediately it occurs, in order to revoke the credentials that depend on it. Selective and efficient revocation is thus facilitated.

Delegation policy is specified in terms of names. An appropriate credential certificate is created for the delegate. Its invalidation can be caused by the signalling of an event, which might be a timer expiry or completion of the task for which delegation was used, as well as through revocation by the delegator.

The scheme extends naturally to support evolution of the entry rules. The proof tree supporting an RMC may be made to depend on the entry rule used during authentication. If that rule is changed, this may be signalled as an event and the RMC revoked. The client must then request a replacement RMC according to the new policy.

The system is inherently distributed and a proof tree may span several systems. If a component fails then superior components that rely on its guarantee of the truth of a predicate must have a policy for dealing with the situation. We have provided a failure detection mechanism based on a heartbeat which may be tuned to arbitrary granularity. A highly sensitive service may suspend the activity of a name that depends on a credential held by a failed system; other services may be more trusting and allow affected names to continue to operate. Both the heartbeat period and the failure policy are service dependent.

The system is inherently decentralised and can support complex trust relationships between services. This is ideal for open distributed environments; no centralised access control scheme is imposed. We have provided a general and flexible mechanism with which complex and subtle policies may be built. Existing mechanisms may be made to interoperate with Oasis.

References

[1] B. W. Lampson,
"Protection", Fifth Princeton Symposium on Information Sciences and Systems, pp.437-443, Princeton University, March 1971, Reprinted in Operating Systems Review, 8(1), pp.18-24, January 1974

[2] Ravi S.Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman
"Role based access control models", IEEE Computer 29(2), pp. 38-47, February 1996

[3] Edward Wobber, Martin Abadi, Michael Burrows and Butler Lampson
"Authentication in the Taos Operating System", TOCS 12(1), pp. 3-32, February 1994

[4] Morris Sloman,
"Policy Driven Management for Distributed Systems",
In *Journal of Network and Systems Management, Plenum Press*, 2(4) 1994

[5] David Ferraiolo, Janet Cugini, and Rick Kuhn
"Role Based Access Control (RBAC): Features and Motivations", Annual Computer Security Applications Conference, IEEE Computer Society Press, 1995

[6] J.A. Bull, Li. Gong, and K.R. Sollins
"Towards Security in an Open Systems Federation". In *Lecture Notes in Computer Science 648.*
ESORICS 92, pp 3-20, Springer Verlag 1992.

[7] R.J. Hayton,
"Oasis, An Open Architecture for Secure Interworking Services",
PhD thesis, Cambridge University, Technical Report 399, 1996

[8] Li. Gong
"A secure identity-based capability system". In *Proceedings of the 1989 Symposium on Security and Privacy*, pp 56–63. IEEE, May 1989.

[9] Sai Lai Lo,
"A Modular and Extensible Network Storage Architecture",
PhD thesis, Cambridge University, January 1994, Technical Report No. TR 326.
Distinguished Dissertations in Computer Science, CUP 1995

[10] J.M. Bacon, J.O. Bates, R.J. Hayton, and K. Moody,
"Using Events to Build Distributed Applications", Proc IEEE SDNE Services in Distributed
and Networked Environments, pp148 - 155, Whistler, British Columbia, June 95