Resource Management in a Distributed System

Mark Yudkin

Cambridge University Computer Laboratory,
Corn Exchange Street, Cambridge CB2 3QG, England.

Abstract

In the Cambridge Distributed System, developed at the Computer Laboratory, Cambridge, terminals, machines and peripherals are separated by a ring network. A problem that arises is the establishment and control of suitable connexions between the individual components, so that users may use machines, machines may use peripherals or other machines, etc. in a secure manner.

This paper describes an experimental mechanism used to control the connexions and provide security in our particular system. The solution is not restricted to the Cambridge ring, although a different architecture may suggest alternative solutions more suited to its specific features. Some possibilities are discussed.

1. Introduction

The Cambridge Distributed System [5] provides a large number of services running permanently at fixed addresses ("static services": those which run on dedicated machines) carrying out particular functions, such as printing, tape handling, authentication, logging and giving the time. There are also a set of machines, known collectively as the Processor Bank; ownership of whose members is uncommitted. They may be allocated to users when required to perform whatever service the user desires. Control of machines in the processor bank is performed through the use of a set of services known as the Ancillae¹ which enable the temporary owner of a machine to exercise at least the same degree of control over that machine as if its control panel were physically accessible to its owner.

1 Ancilla - Latin: housemaid

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

There is a separate Ancilla for each type of processor and the Ancillae provide a uniform interface for all processors.

Any service may use other services, subject to authentication issues. There are also a small number of services without dedicated machines, and when appropriate tools for managing these become readily available the number of such services is likely to increase. It is the secure management of these so-called "dynamic services" which is of interest here; management being defined as controlling the creation and destruction of services. The mechanism needs to be able to handle both dynamic and static services uniformly and transparently: users ought not to be able to tell the difference, except perhaps for timing effects.

We wish to provide a system allowing services of any complexity to be created and layered on top of other services. Security dictates that only those authorized to use a service may do so; allowing mutual cooperation of suspicious services means that keys representing users or services cannot be passed from one to the other, for fear of their being abused.

2. Authentication

Objects (users, machines, services etc.) are represented in the Cambridge Distributed System by their possessing an entry in an Active Name Table. This holds their name, authority, access key and control key, each represented by random 64-bit unique identifiers. The access key is the proof that the owner is the object whose identity is name, where authority is the agency which caused the entry to be created. The control key is required for the entry to be deleted or modified. Further details may be found in [4] or [5].

In our system, however, one object may not wish to apply to a service by passing the objects own access key because it does not trust the service sufficiently. What we do is to interpose a trusted

service (called the Manciple¹) one of whose functions is to perform the relevant authentication (of the client) on behalf of the service. If successful, the Manciple will issue a key allowing access to the service.

Thus a service, upon being created, will issue an allowance (a 64-bit well-known permanent unique identifier); objects suitably authorized will obtain a capability (a 64-bit random number encoding the allowance, his name and certain access rights). Presentation of this capability to the service enables the object to obtain the service with no security compromise. The service can find out who its client is but not that client's access key. This capability is separate from the address, since the address at which a service may be obtained is generally fixed in a "static" service. This is different from Tanenbaum and Mullender's Amoeba [7]. One advantage is that the initial contact address is not subject to negotiation between the allocation system and the newly created service.

In practice, services may offer many different classes of service, depending upon its client's qualifications. For example, one user's access to a filing system generally differs from other users'. In addition, systems programmers generally have greater access to system files, and represent a different class of user. In the authentication system on the Cambridge Distributed System there is the notion of a privilege, being represented by an Active Name Table entry with the authority "PRIV". Objects may claim those privileges to which they are entitled by applying to a Privilege Manager.

In our system, services represent their different classes of service by different allowances. Upon creation, a service issues a list of allowances; associated with each will be a non-empty set of privileges, all of which must be held by the client before that particular allowance may be claimed. (In fact, the allowance list will probably be issued by the service's Secretary: this is discussed further in section 3). The list of allowances is ordered so that if allowance A appears before allowance B, then the set of privileges associated with A is not a subset of the set of privileges associated with B. Furthermore no two (distinct) allowances in a list have identical sets of privileges. These conditions are sufficient to ensure that the list of allowances can be ordered and that the list is finite, given a finite number of privileges. In practice allowance lists tend to be small.

1 Manciple - (sb.) an officer who buys provisions for a college.
More obvious terms (like "resource manager") evoke different pre-determined connotations in different environments.

An operation called refinement is used to issue an allowance to a client. A client's access key may be associated with a number of privileges, making it a simple matter to determine if a client has a particular privilege. Refinement works by scanning the list of allowances in order, checking each allowance in turn to see if the client is entitled to that allowance (that the client holds all of the associated privileges), stopping as soon as a suitable allowance is found.

Since services may be layered on top of other services, one might think that possession of a capability implies possession of the capabilities for all subservices. However, users of a service must not be allowed full access to the subservices. (Imagine the chaos that could arise if users of a large time-sharing operating system were given the same access to the disk drives as the operating system.) Possession of a capability by a service does not imply possession of that capability by its clients. It is worth comparing this with the analogous case of protected procedures in capability computers such as CAP [8] or Hydra/C.mmp [9].

To protect against failures, capabilities have an associated timeout, and must be refreshed at regular intervals. If a capability times out, the reference count associated with the service is decremented. Upon reaching zero, the reference counts for all subservices are decremented and the service can be reclaimed. This is because there may be many capabilities in existence for a service. If a subservice is revoked, so are all services layered on top of it. Since services appear to be layered as a tree structure - a spanning tree of the dependency graph, there can be no cycles.

3. The Manciple and Service Secretaries

At the centre of the resource management system is the Manciple, which looks after all requests for services. The Manciple takes very little action to fulfil requests, it simply forwards requests to other services (known as Service Secretaries) whose purpose is to provide the particular service when required. By causing all requests to be channelled through the Manciple, the authentication issues may be easily handled.

Service Secretaries must be provided for each "generic" service type and these implement all policy issues related to the provision of the service. It is also their job to know how to manufacture the services they manage and to provide the allowance lists. There is only ever one notional instance of a Service Secretary in existence at any time and it must keep its state over failures (the Manciple provides a suitable stable storage facility for Service Secretaries). Distributing a Service Secretary over

several machines is analogous to distributing the database associated with a Secretary.

In the following, items in square brackets ([]'s) are optional, items which may have 0 or more occurrences are followed by an asterisk (*). The reply will contain a return code, which is zero if successful, when the additional information shown will be present, otherwise it should be interpreted as an error code.

3.1 Manciple Interfaces

The Manciple provides the following interfaces for clients:

```
service (service name, caller's access key,
        [caller's control key], caller's name,
        caller's authority, attributes*, [rights],
        privileges*, [uninterpreted bit string])
    → (service address, capability)

refresh (capability, timeout) → ()

finish (capability) → ()

allowPass (capability, other's access key,
          other's control key, other's name,
          other's authority, [rights])
    → (new capability for other)

refine (capability, [rights], like privileges*,
       hate privileges*)
    → (new capability)

amplify (capability, [rights], like privileges*,
       hate privileges*)
    → (new capability)

identify (capability)
    → (service name, service address,
       service secretary's name, owner's name,
       owner's authority)
```

There are four rights associated with a capability: passable, amplifiable, public and unconfined. Passable and amplifiable determine whether or not pass and amplify is allowed; public allows a service the user's access key (and control key if given); unconfined is discussed briefly in section 4. Those operations which return capabilities (allowPass, refine and amplify) will AND together the rights of the original capability with the rights given.

The **service** interface enables the user to obtain a service by quoting its name. He gives the Manciple his authentication information, together with a list of privileges he could hold but may not presently have (the Manciple will check such a claim with the Privilege Manager). For each service there will be a list of attributes he may require: these are clearly service specific. Like all names, they are permanent unique 64-bit identifiers. The uninterpreted bit string may contain further information about the required service; in any case the Manciple simply passes it to the Service Secretary for the service, uninterpreted (see later).

The **refresh** interface implements a dead-man's handle. Because it is difficult to determine if a remote service has died, we require that the owner regularly refresh its capability. In practice, owners generally hand the capability to the service for it to refresh. The timeout is some reasonable value, typically a few minutes. Should the capability time out the owner or service is deemed to have lost interest (or died) (with noone debugging it). When all capabilities associated with a service have timed out or been deleted (see finish below), the resources associated with a service may be forcibly reclaimed.

The **finish** interface allows a service or its owner to indicate that the owner of the given capability has finished with the service. The resources associated with the service may be reclaimed when all outstanding capabilities for the service have been deleted or have timed out.

The **allowPass** interface provides the holder of a capability the ability of making a new capability for another user. It requires the holder (H) to provide the authentication information for the other user (X). The operation will fail if the capability does not have the passable right. The new capability will be for user X, and the **verify** interface (see below) will return X's name rather than H's. If H wishes to pass a capability to user X, and they are suspicious of each other, then they should arrange for the action to be performed by a service which they both trust (H passes the capability to the mutually trusted service while X passes his authentication information to it).

The **refine** interface enables the allowance list from which the capability was derived to be refined again, starting at the allowance corresponding to the given capability.

The **amplify** interface enables the allowance list to be refined again starting at the top, but stopping no later than the given capability. The operation will fail if the capability does not have the amplify right.

The **identify** interface is not actually necessary, since all the information it provides can be determined at the time it was issued. Nevertheless it can be useful to be able to determine the information at a later stage.

The Manciple provides the following interfaces for services or their Secretaries:

```
verify (capability, service's access key,
       service's name, service's authority)
    → (allowance, user's name)

check (capability, service's access key,
      service's name, service's authority) → ()
```

```

retrieve (service secretary's access key,
         service secretary's control key,
         service secretary's name,
         service secretary's authority,
         service secretary's address, allowance)
→ (service address, service's access key,
   allowance list, subservice's capability*,
   [uninterpreted bit string]) *
revoke (capability, service's access key,
       service's name, service's authority) → ()

```

The **verify** interface is provided to allow a service to check that a capability is valid and has not been revoked (see below). It returns the allowance that generated it, together with the owner's name.

The **check** interface is like the **verify** interface above, but returns no additional information.

The **retrieve** interface allows a Service Secretary to retrieve information it previously deposited with the Manciple. This implements state saving stable memory for the Service Secretary. The major item is the uninterpreted bit string, which is the uninterpreted bit string returned by the **instance** request (to be discussed). If allowance is omitted the result is a list of the information normally provided, containing all information previously deposited.

The **revoke** interface enables a service or its Secretary to revoke a capability. This means that any attempt to verify or check it will fail, as will refresh, allowPass, refine, amplify and identify. If the capability is omitted, all capabilities for the given service are revoked.

3.2 Service Secretary Interfaces

The Service Secretary looks, to the Manciple and the rest of the system, like a service. It differs in that its "Service Secretary" is the Manciple, and its only users are the Manciple and the services it provides. Requests from its services include that service's access key for authentication (a service can hardly be suspicious of its Secretary), and requests from the Manciple include the Service Secretary's access key. Since the Service Secretary's access key is known only to itself, the Manciple (which will not disclose it) and the authentication system, it is adequate proof; no capabilities are needed.

Service Secretaries provide the following interfaces for the Manciple:

```

instance (service name,
         service secretary's access key,
         attributes*, [rights],
         [uninterpreted bit string])
→ (service address, service's access key,
   service's control key,
   service's authority, (allowance,
   associated privilege set)*,
   subservice's capability*,
   [uninterpreted bit string])

```

```

finished (service's access key,
         service secretary's access key,
         allowance) → ()

```

The **instance** interface enables the Manciple to obtain a service from its secretary. The service name parameter is superfluous, but enables the service secretary to detect errors in the service - Service Secretary mapping. The attributes list, rights and uninterpreted bit string are those which were passed in to the Manciple's **service** interface. The Service Secretary will create a service instance (possibly calling the Manciple's service interface for required subservices). The result will be the service's address and authentication information (its name is known by the Manciple - it was part of the request), together with the allowance list and capabilities for all subservices used in creating the service. The uninterpreted bit string contains state information which the Secretary wants kept stably. It can be retrieved by the **retrieve** entry (see above).

The **finished** interface instructs the Service Secretary to clean up after the service instance, as it is no longer required. The Service Secretary does not need to advise subservices used in making the service, as the Manciple does that.

There are no standard interfaces Service Secretaries must provide for their services; any interfaces which are provided are at the discretion of the author of a service and are private.

4. Other Possibilities

The mechanism used is that chosen for the Cambridge Ring. It is obviously possible to use the same mechanism on other local area networks. However some features of the Cambridge Ring have affected the design. The Cambridge Ring has no broadcast mode. The ring interfaces for many of the processor bank machines contain small awkward micro-processors which are unsuitable for anything but trivial ring connexions. The ring is assumed to provide secure point-to-point transmission: transmissions cannot be compromised or observed by any processor (without physically altering the ring hardware).

If the ring interfaces contained more powerful processors, much of the allocation and authentication system could be put into the interfaces. Because the interfaces run fixed, unchangeable programs, we have user protected memory to use (as with many operating systems) for holding classified information. Therefore access keys could be held in the interface's memory, and the capability mechanism introduced here for dealing with mutual suspicion would be largely unnecessary. This is now comparable to a machine with a capability architecture, such as CAP [8] or

Hydra/C.mmp [9], where each domain is a remote service. The Manciple would have only to handle the book-keeping. Services running in the host machine would refresh the access key by sending refresh messages to the host's interface, which would perform the refreshing across the ring. This however makes it very hard to maintain and enhance the system and is probably unsatisfactory for these reasons alone.

If the ring were to have a broadcast mode, the book-keeping facilities of the Manciple could be supplemented by broadcasting messages to every machine to find out what it was doing. The ring interfaces would always be able to answer such enquiries by remembering what was loaded (loading requests go directly to it); failure to provoke any reply indicating failure of the node. The stable memory requirement of the Manciple could be removed.

The total system could therefore comprise a map between generic service types and their Secretaries, and a number of Service Secretaries which upon creation insert their addresses into the map. The presence of a unique instance of a Service Secretary is ensured using the same mechanism as is used in a local area network in which any station can become the monitor. A booting service would also be provided to boot Services Secretaries and services, although this could be removed if the interfaces were capable of fulfilling booting requests directly. Taken to the extreme, interfaces could implement the entire allocation scheme.

Another issue is the data confinement problem. Although it is clear that the partial confinement problem cannot be solved without specialized architectures for each processor on the network [1,2,3], it is possible to solve the total confinement problem of preventing a single user service from accessing services other than those explicitly allowed by its client. In this way data can be kept confined to a specified set of services, which is sufficient for some classes of problem. In particular those services which do not require other (suspicious) services can be prevented from disclosing any information sent to them. This requires the notion of transmission keys which need to be handled by the interfaces; interfaces refuse to transmit a block unless presented with a valid transmission key for the particular transmission by the host. Further details will not be given here.

If the network could not provide secure end-to-end transmission, some form of end-to-end encryption, such as the Needham and Schroeder protocols [6] could be used.

Of these ideas, only data confinement has been implemented, and this cannot be enforced by every processor in our environment.

5. Performance Measurement and Conclusion

The Manciple has been implemented on the Cambridge Ring and together with a couple of Service Secretaries is capable of providing a "Tripos" operating system or dialect running on LSI4 or 68000 microprocessors. A simple user interface is provided for the convenience of users. Some performance measurements have been taken, performance being measured mainly as the time taken to provide the required service. These will be summarized here.

In general the mechanism works satisfactorily. To reduce the number of physical machines needed, the Manciple and Service Secretaries live on the same machine, although their only communication is through the ring. The system has been demonstrated with the Manciple and Service Secretaries running on separate machines.

The dominant factor in performance is the time taken to load a machine. With a 10MHz packet ring containing 3 packets (allowing use of every fourth) of 38 bits (16 data bits), plus a 1 bit (minimum) inter-packet gap, maximum point-to-point bandwidth is 1Mbit/sec. Booting a machine involves the data travelling around the ring twice: once from the fileserver to the Ancilla and once from the Ancilla to the machine. Ignoring protocol and processing overheads, loading 64Kbytes takes 1 second. In practice there are delays introduced by processing in the fileserver and Ancilla, reading the disk, ring contention, etc. and the actual loading time is about 3 - 5 seconds (depending upon load).

Access to the file server is the main bottleneck. Thus the decision to have the Manciple provide the Service Secretaries' stable storage, whilst a little unclean, appears to have been correct compared with the alternative of having each use the file server directly. The Manciple backs up its state using the file server. The cost of non-loading ring transmissions, authentication checking and processing in the Manciple and Service Secretaries appear negligible by comparison. There are occasional delays seen when the fileserver is loaded, since the Manciple will not verify or check a capability before the associated state has been saved: it gives a retryable error code.

It seems likely that a considerable gain in performance could be achieved by pre-manufacturing commonly used services during idle moments. Should a different service be required no harm has been done. The Manciple would have to be responsible for record keeping, and either it, or another service could be responsible for predicting future demand, balancing the numbers of services and other decision making. It would be useful for Service Secretaries to provide an interface of the form: "If a service instance had been

requested, would it have been provided?" This interface is designed to provide a hint only; its purpose is to allow a possible balance between services to be determined rapidly, although not perfectly. An affirmative answer will continue to be correct at least until the Manciple makes a service request (assuming no hardware failures). Pre-manufacturing has not been implemented in the current system.

References

1. D.E. Denning
A Lattice Model of Secure Information Flow
Comm. ACM 19/5 (May 1976) pp 236-243
2. D.E. Denning & P.J. Denning
Certification of Programs for Secure
Information Flow
Comm. ACM 20/7 (July 1977) pp 504-513
3. J.S. Fenton
Information Protection Systems
Ph.D. Thesis, University of Cambridge 1973
4. C.G. Girling:
Authentication in Project Universe
Proceedings 6th International Conference
on Computer Communication pp 395-400
5. R.M. Needham & A.J. Herbert:
The Cambridge Distributed Computer System
Addison Wesley Publishing Company 1982
6. R.M. Needham & M.D. Schroeder
Using Encryption for Authentication
in Large Computer Networks
Comm. ACM 21/12 (Dec 1978) pp 993-999
7. A.S. Tanenbaum & S.J. Mullender
An Overview of the Amoeba Distributed
Operating System
ACM Operating Systems Review July 1981 pp 51-64
8. M.U. Wilkes & R.M. Needham
The Cambridge CAP Computer and its
Operating System
North Holland 1979
9. W.A. Wulf, R. Levin & S.P. Harbison
Hydra/C.mmp: An Experimental Computer System
McGraw Hill 1981