A SECURITY SCHEME FOR RESOURCE SHARING OVER A NETWORK

John Burns
Chris J. Mitchell

Hewlett-Packard Laboratories
Filton Road
Stoke Gifford
Bristol BS12 6QZ
England

16th October 1989

ABSTRACT

The purpose of this paper is to describe a security scheme for a special-purpose resource-sharing system for networked computers. The scheme makes use of cryptographic constructs called *coupons*, issued by a central authority, and representing the right to use a certain amount of resources on a specified machine. The security scheme is described in detail, and an analysis of its security is also given.

1. INTRODUCTION

The purpose of the scheme described here is to provide security for a special-purpose resource allocation scheme. We suppose that, in a network of computers, two conditions exist:

- spare resources are available, i.e. there exist computers on the network which do not use all their available CPU time.
- jobs exist which require more computational effort than can be provided by a single machine in a reasonable amount of time.

We then suppose that the owners of the under-used machines are willing for their unused resources to be utilised by other users with resource shortages, as long as these other users are appropriately authorised.

In this paper we describe a scheme for providing such authorisation information. This system uses the notion of a *broker*, who keeps details of all machines with spare resources, and allocates these resources to other users who need them. This broker must be trusted by all the entities within the network which is serves. We suppose that the computers on the network may be either *users* or *suppliers* of resources (or both). A *supplier* makes the offer of unused capacity to the broker. The broker then allocates these resources to *users* who request them from the broker.

The protocol described below bears some resemblances to the Kerberos authentication protocol described by Steiner et al., [11]. However there are a number of significant differences. From a cryptographic point of view the most significant is that, unlike Kerberos, the protocol described here is independent of time-stamps, and hence does not rely on synchronised clocks. In addition, within its designed application it introduces no additional messages, and hence its overheads on the underlying communications system are very low.

2. THE PROTOCOL

2.1 Notation and assumptions

The solution described in this paper relies on the use of conventional (symmetric) cryptography, as typified by the DES block cipher algorithm, [1], [6]. We assume that each user and supplier in the network shares a secret key with the broker, and that this secret key corresponds to an algorithm capable of being used both for data encryption (for confidentiality) and for computation of Message Authentication Codes (MACs) for data integrity and authentication. We denote the key shared by network entity E and the broker by K(E). Note that different versions of this key should be used for encryption and MAC computation; for example, the key could bit-wise exclusive or-ed with a fixed 'mask' (not all zeros or all ones) when used for MAC computation.

We denote by

$E_K[I]$

the encryption of the data I using key K, and we write

$M_K[I]$

for the MAC computed on the data I using the key K. The difference between these two concepts is that knowledge of $E_K[I]$ and K enables the recovery of I after the application of the appropriate decryption function, whereas $M_K[I]$ is a short, fixed length 'checksum' enabling deliberate or accidental modifications to data to be detected.

An example of a suitable algorithm is provided by the DES (or, for that matter, any other block cipher algorithm). Data encryption could be achieved by using DES in the standardised Cipher Block Chaining (CBC) Mode, [2], [7], [8], and the MAC computation could again be based on DES in CBC mode; see, for example, [3], [4], [9].

2.2 Resource tickets and their management

Fundamental to the operation of the proposed system is the list of information on available resources kept by the broker. This list consists of a series of *tickets*, issued by resource *suppliers* on the network. Each ticket contains the following three items of information: the name of the supplier, a supplier serial number and a value parameter (giving an indication of the amount of resources being offered). This value parameter may, for example, indicate an upper limit on CPU time, the type of processor involved and/or an upper limit on available RAM; the precise use of this parameter is beyond the scope of this paper and will, in any case, be very dependent on the particular implementation of this scheme.

Each supplier will generate one or more of these tickets (possible of varying values) and send them to the broker in protected form. As and when the tickets are eventually used (as we describe below), and as further unused resources become available, so the supplier generates more tickets and supplies them to the broker. Serial numbers are allocated by the supplier, and it is important that the supplier ensures: (a) that no two tickets with the same serial number are issued, and (b) that a record is kept of the serial numbers of outstanding (i.e. unused) tickets. This is necessary in order to prevent re-use of old tickets.

More formally, a ticket issued by supplier S has the form:

$$(S, N, V_t)$$

where N is the serial number, V_t is the value and the use of commas denotes concatenation of data. The three items of information within the ticket are precisely the items of information stored within the broker. When the ticket is shipped from the supplier to the broker it has the form:

$$(S, N, V_t, M_{K(S)}[S, N, V_t])$$

i.e. a MAC on all the data within the ticket is appended to the end of the ticket. This MAC is computed using the key

shared by S and the broker. The use of this MAC enables the broker to check the validity of the ticket.

Before proceeding we consider the deletion of stored tickets. As we described above, lists of tickets will need to be stored both by ticket suppliers and the broker. The broker deletes a ticket once it has been allocated to a particular user (as described in 2.3 below). The ticket supplier deletes a ticket when a user wishes to use the resources specified in it (see 2.4 below). However, in certain circumstances, neither of these types of event will occur. For example, the broker may never issue a ticket because of a shortage of users, and a user may not need to use a resource requested from a broker, and hence the supplier will never get a request for resources corresponding to one of its stored tickets.

For this reason both the broker and the supplier will automatically delete tickets from their stores after a specified time interval (depending on the type of network involved). At worst this will have the effect of meaning that, occasionally, messages from users to suppliers will be rejected because the corresponding tickets have expired and been discarded. To minimise the probability of this occurring it is probably wise for the broker to keep tickets for a shorter time than the supplier.

In many applications it may be desirable for suppliers of tickets to specify the lifetime of their tickets. Details of how this may be achieved with only a small modification to the basic protocol are given in 4.3 below.

2.3 Resource requests and coupons

We now consider how users request resources from the broker. Suppose user U wishes to make use of spare resources on the network. U issues a *request* to the broker, which contains the following two items of information: the name of the user and

a value parameter, V_r, giving an indication of the amount of resources required. As with the value parameter in supplier tickets, the precise nature of the request value parameter is beyond the scope of this paper. These requests are sent in protected form as follows:

$$(U, V_r, M_{K(U)}[U, V_r])$$

i.e. a MAC on all the data within the request is appended to the end of the request. This MAC is computed using the key shared by U and the broker. The use of this MAC enables the broker to check the validity of the request.

On receipt of a request (given that the MAC check proves it to be valid) the broker will compare it with the outstanding tickets, and decide which of the tickets are to be allocated to the requesting user. This will be done using a process which might take into account the following: the privileges of the requesting user, the size of the value in the request and the number and values of the outstanding tickets.

For each ticket allocated to the requesting user, a *coupon* message is generated and sent (in protected form) to the requesting user. If the ticket being allocated to user U has the form:

$$(S, N, V_t)$$

then the corresponding coupon message has the form:

$$(S, N, V_t, U, M_{K(S)}[S,N,V_t,U,SK], M_{K(U)}[S,N,V_t,U,SK])$$
$$(E_{K(U)}[SK]),$$
$$(E_{K(S)}[SK])$$

The *coupon* itself is the first part of the message, namely:

$$(S, N, V_t, U, M_{K(S)}[S,N,V_t,U,SK], M_{K(U)}[S,N,V_t,U,SK])$$

where SK is a *session key* randomly generated by the broker and unique to each coupon. Also sent with the coupon are: a copy of the session key, SK, encrypted under the secret key shared by the user and the broker:

$$(E_{K(U)}[SK])$$

and a copy of the session key, SK, encrypted under the secret key shared by the ticket supplier and the broker:

($E_{K(S)}[SK]$)

When the broker sends such a coupon, the corresponding ticket is deleted from its list. For each coupon (and accompanying keys) received by the user, the following procedure is followed:

- the copy of the session key SK encrypted under K(U) is decrypted and SK is recovered.
- the appropriate MAC on the coupon is then checked using K(U) and the recovered value of SK.
- if the MAC authenticates the coupon, then the coupon is stored ready for use, together with two other pieces of information: the session key for the coupon (SK), and the session key encrypted under the supplier's secret key (as provided by the broker):

  ($E_{K(S)}[SK]$).

2.4 Resource supply

When the user receives the coupons from the broker, it is then up to the user to divide the task to be performed into suitable pieces corresponding to the values in the coupons. We now consider the process followed when a user wishes to use a coupon.

The user, U, sends the coupon to the named supplier, S, (omitting the MAC computed using K(U) as this is of no use to S), together with two other pieces of information:

- first, U also sends S a copy of the session key for the coupon encrypted under S's secret key (as provided by the broker):

  ($E_{K(S)}[SK]$).

- second, U sends all the necessary information about the task U wishes S to perform (probably including all the necessary executable code) authenticated using SK, i.e. if the task information is T then U sends S:

  (T, M_{SK}[T]).

U also stores information about the particular task T, together with the values S and N, so that, when the results of performing T are returned to U by S, they can be matched to T.

When S receives the coupon and the associated task information, S follows the procedure below:

- the copy of the session key SK encrypted under K(S) is decrypted and SK is recovered.
- the MAC on the coupon is checked using K(S) and the recovered value of SK.
- the MAC on the task T is checked using SK.
- if the MACs authenticate the coupon and the task to be performed, and the serial number N corresponds to an unredeemed serial number stored within S, then the task T is executed.
- the ticket serial number, N, in this request is deleted from the list of outstanding tickets.

When the given task has terminated, the results of performing the task are returned to U in a protected form. More specifically, if R represents the results of performing the task T, then S returns to U the following:

(R, S, N, M_{SK}[R, S, N])

where the name of S and the serial number N identify uniquely to U which task T these results correspond to. This completes the description of the system's operation.

3. ANALYSIS OF THE PROTOCOL

A system such as the one described in the above section may be subject to a variety of attacks by unauthorised third parties wishing to misappropriate resources. We consider some of these attacks, and examine how the system resists them.

Before proceeding note that the system described does not attempt to provide any confidentiality services for the tasks distributed around the network. Rather, the scheme is designed to protect suppliers against misappropriation of their resources. However, if they were ever required, confidentiality mechanisms could probably be added to the above protocol without too much difficulty.

3.1 Replay attacks

Every message in the protocol described above involves the use of authentication checks (MACs) computed using secret keys. Therefore, given the MAC algorithm is sound, construction by unauthorised users of completely spurious messages is impossible unless keys become compromised (we discuss this latter possibility in 3.2 below). Thus, apart from key compromise, the only possible attacks involve some form of replay.

We now examine in turn the effects of replaying each type of message in the system. There are essentially five types of message in the above system: tickets sent from S to the broker, requests sent from U to the broker, coupons sent from the broker to U, coupons sent from U to S and results sent from S to U.

When transmitted from S to the broker, a ticket has the form:

$$(S, N, V_t, M_{K(S)}[S, N, V_t])$$

and all the data in the ticket is protected by a single MAC. Therefore, in this case the only possibility is to replay the

message unchanged. If such a replay occurs, all that will happen is that the broker will store a duplicate ticket, and possibly issue a duplicate coupon to an unsuspecting user. If this does occur the only harmful end result is that one of the two recipients of the duplicated coupon will have their request refused by the supplier because the ticket has already been used. This is a minor problem and does not breach the security of the system.

When transmitted from U to the broker, a request has the form:

$$(U, V_r, M_{K(U)}[U, V_r])$$

and all the data in the request is protected by a single MAC. Therefore, as before, the only possibility is to replay the message unchanged. The end result will be that U will be given coupons for which U has no real use, and perhaps some of these coupons will be wasted. A persistent attacker of the system could repeatedly do this, with the aim of diverting all the coupons to one user and thereby preventing their allocation to genuine users. The obvious way to alleviate the effects of such an attack is to restrict the percentage of coupons which may be allocated to any one user. However, the main thing to note is that this attack would not compromise the basic integrity of the system, since no resources would be allocated to unauthorised users. It has to be recognised that 'denial of service' attacks can always be launched against such systems, in the extreme simply by disrupting the communications between end users of the network.

When transmitted from the broker to U, a coupon message has the form:

$$(S, N, V_t, U, M_{K(S)}[S,N,V_t,U,SK], M_{K(U)}[S,N,V_t,U,SK])$$
$$(E_{K(U)}[SK]),$$
$$(E_{K(S)}[SK])$$

This message has three distinct parts; however all three parts are 'bound together' by the value of SK, which is unique to this particular coupon. Therefore, the only possibility is to replay the message unchanged. The end result of such a replay

will be that U ends up with two copies of the same coupon; if U tries to use both of them then the second will be rejected by the supplier S. This again does not represent a major hazard to the security of the system, since such events are bound to occasionally occur because of the automatic deletion of tickets by suppliers.

When transmitted from U to S, a coupon message has the form:

$$(S, N, V_t, U, M_{K(S)}[S, N, V_t, U, SK]),$$
$$(T, M_{SK}[T]),$$
$$(E_{K(S)}[SK])$$

This message has three distinct parts; however, just as before all three parts are 'bound together' by the value of SK, which is unique to this particular coupon. Therefore, the only possibility is to replay the message unchanged. The end result of such a replay will be for S to receive two copies of the same coupon from U. The second will be rejected by the supplier S, and so this attack again does not represent a major hazard to the security of the system.

When transmitted from S to U, a results message has the form:

$$(R, S, N, M_{SK}[R, S, N])$$

and all the data in the message is protected by a single MAC. Therefore, in this case the only possibility is to replay the message unchanged. This will mean that U gets two copies of the results of performing the requested task. The second will be ignored since the two copies will be identified as such by the repetition of N (the serial number).

3.2 Deletion of messages

The only obvious effect of deleting any of the messages in transit is to prevent the use of a ticket issued by a supplier, S. S will then be left with an unused ticket in its list. This could happen anyway if a user U never cashes in a coupon issued by the broker. The simplest solution to the

problem of accumulating unclaimed resource tickets is for all suppliers to discard unused tickets within a certain time interval of their issue, as described in 2.2 above.

3.3 Use of cryptanalysed session keys

It will be apparent that the protocol described above bears many similarities to the protocol described in Needham and Schroeder, [10]. Unfortunately, this latter protocol is vulnerable to a certain special kind of replay attack if the confidentiality of a session key is ever compromised; see, for example, [5].

The main difference between the protocol described here and the Needham/Schroeder protocol is the use of serial numbers, which prevent re-use of coupons. This is a great advantage since it also prevents the kind of attack possible on the Needham/Schroeder protocol. We now describe the potential attack in a little more detail.

Suppose that an interceptor, C say, of a coupon has, by some means, been able to discover the session key, SK, used to authenticate the coupon. If the interceptor wishes to use this information to steal resources from the supplier of the corresponding ticket, then a message of the form:

$$(S, N, V_t, U, M_{K(S)}[S, N, V_t, U, SK]),$$
$$(T', M_{SK}[T']),$$
$$(E_{K(S)}[SK])$$

must be constructed and sent to S, where T' is the task that C wishes S to perform.

The first and third parts of such a message can never be computed by C, since constructing them requires knowledge of the key K(S) which we must assume remains secure; of course, if this key was compromised then the entire system would be rendered insecure. Hence the only option for C is to copy these two parts from an observed message and forge the middle

part (containing T'). This would work (with C impersonating U) but it would only work once, since S will delete the ticket with serial number N from its list the first time such a message is received.

In summary, compromise of the session key SK belonging to a ticket will only compromise the security of the resources allocated to that ticket, and will not allow any other resources to be stolen. This is as much as one could expect from a system of this type.

3.4 User attacks

In our discussion above we have considered the case where an unauthorised third party wishes to steal resources from a supplier. We conclude this analysis by considering the case where a valid user wishes to try and obtain more resources than are allocated by the broker.

As in 3.3 above, such a user U must send a message of the following form to a supplier S:

$$(S, N, V_t, U, M_{K(S)}[S, N, V_t, U, SK]),$$
$$(T', M_{SK}[T']),$$
$$(E_{K(S)}[SK])$$

However, user U is in no better a position than the third party C described in 3.3 above to forge the first or third parts of such a message. It is therefore not possible for a user to obtain resources not allocated to it (unless the cryptographic functions used are insecure or secret keys are compromised).

4. POSSIBLE EXTENSIONS

There are many ways in which the above protocol could be extended to provide additional facilities. We consider three such extensions here.

4.1 Multiple results messages

The protocol described above allows for the resource supplier, S, to return a single results message to the user, U. In some circumstances (particularly if the task being undertaken by S is a long one) it would be desirable to allow S to return intermediate results messages.

Currently the defined protocol will only allow the transmission of one such results message - all subsequent messages will be rejected as replays. To modify the protocol to allow multiple results message requires S and U to store and use an additional serial number, P_N say. The form of the results message will then be

$$(R, S, N, P_N, M_{SK}[R, S, N, P_N])$$

In the first results message P_N is set to 1, and is then incremented for each subsequent results message. U will only accept these messages if the new value of P_N is strictly larger than the previously received value for this particular value of N. The use of this serial number will prevent replay attacks.

4.2 Splitting tickets

When the broker receives a ticket from a supplier S, the value in the ticket may be large compared with the values of coupon the broker is being requested to issue. In such circumstances it would be desirable if the broker could divide the ticket into two or more parts and issue coupons whose values sum to the value of the ticket. One way in which this might be

achieved securely is as follows.

When the broker issues a coupon, (representing part of the value of the ticket issued by supplier S with serial number N and value V_t), an additional serial number C_N is included. The number C_N is initially set to 1, and subsequently incremented every time a new coupon is issued representing part of the same ticket. The issued coupon will then have the form:

(S,N,C_N,V_C,U, $M_{K(S)}[S,N,C_N,V_C,U,SK]$, $M_{K(U)}[S,N,C_N,V_C,U,SK]$)

where V_C represents the value of the coupon and is not more than the value given to the issued ticket (V_t). The broker must ensure that the sum of the coupon values V_C issued against a ticket never exceed the value of the ticket (i.e. V_t).

When U receives a coupon and uses it to issue a request, U not only stores information about the task T and the values S and N, but also stores the value C_N. This value is used to help match received results messages against stored requests. When U requests S to perform a task, the communication is just as described in 2.4 above, except that the coupon shipped from U to S now contains the value V_C.

The ticket supplier, S, is also required to store additional state information, namely: the initial value of the ticket V_t, the value so far consumed (i.e. the sum of the values V_C of the coupons so far received bearing the serial number of the ticket), and the values for the serial numbers C_N so far received. When a coupon is received two additional checks are performed: the coupon number C_N is checked against the stored values for this N to detect replays, and the value on the coupon, V_C, is added to the value so far consumed, and a check is made to see that the total value does not exceed the value originally assigned to this serial number N.

Finally note that the only other change to transmitted messages is in the form of the results message, which also

includes the new serial number C_N, and has the form:

(R, S, N, C_N, M_{SK}[R, S, N, C_N])

4.3 Limiting the life of tickets

The automatic expiry of tickets was discussed in 2.2 above. However, as mentioned in 2.2, in some circumstances different suppliers may wish to assign different (shorter) life-times to their tickets. For example a supplier may have resources available for a strictly limited period of time, and may wish to specify that, after the expiry of this time period, the ticket should not be issued.

Of course, one simple strategy would be for suppliers to delete the tickets themselves once the resources have ceased to be available, regardless of the fact that the broker might still issue a corresponding coupon to a user. All that would happen is that the user's request for resources would be denied. However, a more efficient solution might be to include a life-time interval inside each ticket. The general form of a ticket would then be:

(S, N, V_t, T)

where T indicates the length of time that the ticket should be kept by the broker. If the ticket remains unused after time T has elapsed, it is automatically deleted by the broker. Suppliers would normally keep tickets for a slightly longer time interval (as previously discussed).

One advantage of this scheme is that it does not require synchronised clocks in order to operate correctly. All it requires is that the broker's and supplier's clocks run at roughly the same rate (a very reasonable requirement).

REFERENCES

[1] ANSI X3.92, *Data encryption algorithm*, American National Standards Institute (New York), 1981.

[2] ANSI X3.106, *Data encryption algorithm - modes of operation*, American National Standards Institute (New York), 1983.

[3] ANSI X9.9, *Financial institution message authentication (wholesale)*, American Bankers Association (Washington, DC), August 1986.

[4] ANSI X9.19, *Financial institution retail message authentication*, American Bankers Association (Washington, DC).

[5] D.E. Denning and G.M. Sacco, 'Timestamps in key distribution protocols', *CACM* **24** (1981) 533-536.

[6] FIPS PUB 46, *Data encryption standard*, Federal Information Processing Standards Publication 46, National Bureau of Standards, U.S. Department of Commerce (Washington, DC), January 1977.

[7] FIPS PUB 81, *DES modes of operation*, Federal Information Processing Standards Publication 81, National Bureau of Standards, U.S. Department of Commerce (Washington, DC), December 1980.

[8] ISO 8372, *Information processing - Modes of operation for a 64-bit block cipher algorithm*, International Organization for Standardization, 1987.

[9] ISO Draft International Standard (DIS) 9797, *Data cryptographic techniques - Data integrity mechanism using a cryptographic check function employing an n-bit algorithm with truncation*, International Organization for Standardization,

1988.

[10] R.M. Needham and M.D. Schroeder, 'Using encryption for authentication in large networks of computers', *CACM* **21** (1978) 993-999.

[11] J.G. Steiner, C. Neuman and J.I. Schiller, 'Kerberos: An authentication service for open network systems', in: *Proceedings of the USENIX Winter Conference, Dallas, 9-12 February 1988*, USENIX, 1988, pp.191-202.