An Authentication Scheme

for Use in

Dynamic Load Balancing

Colin Parris

Report No. UCB/CSD 87/375

October 1987
PROGRES Report No. 87.7

Computer Science Division (EECS)
University of California
Berkeley, California 94720

An Authentication Scheme for Use in Dynamic Load Balancing

Colin Parris

Master's Project
Computer Science Division, EECS
University of California, Berkeley

ABSTRACT

This report discusses the design, implementation, and performance evaluation of an authentication scheme for use in a dynamic load balancing environment. Measurement experiments were conducted on our implementation using trace-based artificial workloads. The performance gains due to authenticated load balancing, as well as the impact of authentication on the performance gains of unauthenticated load balancing were evaluated. With the mean job response time as the primary index, the results of our study show that authenticated load balancing provides a non-negligible performance gain when compared to a non-load balanced system. As was expected, the performance gains due to authenticated load balancing are substantially smaller than those experienced with unauthenticated load balancing.

September 13, 1987

This work was partially sponsored by AT&T Bell Laboratories and by the National Science Foundation under grant DMC-8503575.

An Authentication Scheme for Use in Dynamic Load Balancing

Colin Parris

Master's Project
Computer Science Division, EECS
University of California, Berkeley

1. Introduction

With the commonplace presence of networks and the compulsion to use every iota of computing resources available, load balancing is fast becoming a deeply felt necessity in distributed environments. Load balancing is the process by which the load on a network of computers is periodically redistributed, thereby achieving a state in which the load on each individual computer is, on the average, approximately the same as the load on every other individual computer comprising the network. The many virtues of load balancing systems have been extolled in the literature, for example by [Barak84, Ezzat86, Hagmann86, Hwang82, Linvy82, Zhou86]. Measurements taken on prototype load balancing implementations support the theory that load balancing is very effective in reducing the mean and the standard deviation of job response times [Linvy82, Zhou86].

With regard to system performance, the worthiness of load balancing in a distributed system has been clearly established. There are, however, some concerns of the computing community that must be addressed before widespread use of load balancing becomes a reality. The foremost concerns are those of security, accounting, and task regulation.

The first concern, that of security, refers to the use of the system's resources by unauthorized users. This unauthorized use usually takes the form of "system penetration", in which an unauthorized user accesses confidential data or utilizes the system's computing power. Insecure systems are also vulnerable to malicious tampering with kernels and utility functions by unauthorized users. Accounting is another major concern, as the system overhead in supporting the bookkeeping for remotely executed jobs appears to be quite large. In load balancing, the bookkeeper must accommodate for the varying costs of CPU time on the different machines that may be present in the load balancing cluster of machines. The user may not wish to incur the additional charge if a relatively trivial job, belonging to that user, executes on a remote machine on which the CPU time is expensive. The job may have been forced to run on this "costly" remote machine if this machine is less loaded than the local machine. The third concern, task regulation, encompasses the problems encountered when owners of machines on which remote jobs are being executed wish to reclaim exclusive use of their machines. At this point, the remotely executing job must either be killed or migrated. Either option, process killing or process migration, lowers the performance of the load balancer by imposing a greater overhead on the system.

In this writer's opinion, the concern of security is the most pressing of the aforementioned concerns. In insecure systems, bookkeeping becomes a trivial concern, as it is not too difficult to avoid charging an account that does not exist or an account undeserving of the charge. Without security, the concern of task regulation may be amplified due to the larger number of remotely executing tasks in the system, as this number also includes the unauthorized ones. This amplification is also due to the placement of unauthorized tasks in the system. Since an unauthorized user may attack the remote host directly, some unauthorized tasks may not be optimally placed and would cause a degradation in load balancing performance. From the arguments presented above it is clear that security issues are of utmost priority and must be addressed first if the use of load balancing is to become universal.

This report addresses one aspect of the security concern, that of authentication. Authentication, in a load balancing environment, refers to verifying, at a remote machine, the identity of a local user whose remotely executable jobs wish to execute on that remote machine. In this report, an authentication scheme for a load balancer operating in a UNIX†-based, local area network (LAN) environment is proposed, implemented, and analyzed. This authentication scheme is

† UNIX is a trademark of AT&T Bell Laboratories.

implemented on a cluster of diskless SUN-2 and SUN-3 workstations running under the SUN/UNIX operating system, supported by a file server, and connected by an Ethernet.

This report is organized as follows. Section 2 presents a survey of work related to this project. The goals, the assumptions, and an overview of the authentication scheme are given in Section 3. Section 4 describes the load balancer in which the scheme is to be implemented, and gives a detailed description of the implementation. The design of the measurement experiments, the results, and the analysis of these results are presented in Section 5. Section 6 summarizes the scheme and its results.

2. Related Work

This section presents a survey of the current, pertinent literature on load balancing and on remote authentication. In a survey of the load balancing literature, the scarcity of actual load balancing implementations is apparent. Of the few load balancing schemes implemented and reported on, the best known are [Hwang82, Hagmann85, Bershad85, Ezzat86, Zhou86].

The earliest of these load balancing schemes was implemented at Purdue University in a Unix environment [Hwang82]. In that implementation, load balancing versions of compilers, assemblers, and text processing programs were built. These load balancing versions called a system scheduling routine, *rxe*, to determine the "lightly" loaded destination host appropriate for execution.

The next notable implementation was done at Xerox PARC, and was targeted to a workstation environment [Hagmann85]. In that implementation, Process Servers supported a collection of personal workstations. These Process Servers may either be permanently dedicated compute servers or workstations donated by their owners when not being used. A central agent is used to collect load information and perform job placements for the entire system.

A load balancer for the Berkeley Unix 4.2 BSD operating system was implemented by Bershad [Bershad85]. In Bershad's implementation only a few programs with large CPU time demands were considered for remote execution. Also, the remotely executed program and its data files had to be explicitly moved to the execution host due to lack of a distributed file system.

One of the more recent load balancer implementations is that created for the NEST system at AT&T Bell Laboratories [Ezzat86]. This load balancer is implemented on a number of workstations connected by an Ethernet-like local-area network. In this implementation, the name of a special program, *rexec*, must be input by the user as a prefix to any command string eligible for remote execution.

To this writer's knowledge, the most recent notable implementation is that reported by [Zhou86]. In this implementation, the load balancer is transparent to the user. The user has the ability to select beforehand the machines and the eligible commands, although default lists of machines and commands are also provided. No modifications to commands and applications are needed in this implementation.

An examination of the load balancing implementations presented above reveals that the issue of security in load balancing environments has not yet been addressed. A review of the pertinent literature pointedly affirms the absence of security considerations. In some of the implementations presented above, another pertinent concern is that the introduction of load balancing into a previously highly secure system may reduce the overall security of that system. An example of this reduction in security can be found in [Zhou86]. In that scheme, an unauthorized user, using the login name of an authorized user, can send a job request to the load balancing module, and that request will be granted. This situation can allow the unauthorized user to obtain access to previously unaccessible accounts and resources, which were formerly accessible to the authorized user only.

In most cases, as the complexity of a system increases, the security concerns of the system increase correspondingly. This situation occurs as the increase in complexity introduces additional security "loopholes" to the system. The introduction of load balancing into most current systems would certainly be classified as an increase in the complexity of the system, thereby suggesting the possible emergence of security problems. Given the concerns presented in this section and in the previous section, the scheme proposed in this paper is aptly justified.

Our scheme focuses on the authentication aspect of security. Therefore, authentication of remote processes in a distributed workstation environment constitutes the category under which previous work related to this scheme can be found. The most relevant work in remote authentication has been reported in the literature by [Birrell85, Taylor86, Joy83, Anderson87].

In the Secure RPC implemented at Xerox PARC, authentication is achieved by the use of a private/public key encryption scheme. In this scheme the security principals communicate with a trusted authentication server and obtain a unique authenticator. This authenticator identifies the principal and contains the conversation key to be used by the two principals for the duration of their conversation. With each authenticator request to a new host, public/private key encryption must be employed, thus increasing user response times. All private and public keys are stored in a Grapevine [Birrell82] distributed database, hence the keys of both of the principals may not be known by any single Grapevine host. In this case, a Grapevine host may need to securely communicate to another Grapevine host, thereby adding further increases to user response times. The entire security of the scheme depends on the security of the authentication service's database.

A scheme for remote authentication has been proposed and implemented by SUN [Taylor86]. At this time, the implementation is used primarily to support remote logins in a distributed workstation environment. In the SUN scheme, the user's password is used to decrypt the private key of the user's public/private key pair. Once the private key is decrypted, it is stored in a keyserver which saves it for use in future RPC transactions. This private key is used to authenticate and support user's requests to remote machines or network services. This authentication is achieved using SUN's *Authentication Protocol.* In this protocol, conversation keys are exchanged that are used during the ensuing conversation. In order to initialize a conversation, the client and the server both correspond with their local keyservers, who then make the conversation keys known to the server and client, respectively.

In the UNIX system a remote process can be invoked by using the *rexec* system call [Joy83]. In this system call, authentication is achieved by having the user supply the username and password. This information can be supplied as parameters to the call or can be supplied interactively. Each call requires the submission of the user's password.

The final authentication scheme to be examined is that employed by the DASH system [Anderson87]. In this scheme, kernels are authenticated using private/public key encryption and a secure authentication channel is formed between these kernels. If user A on machine M1 wishes to communicate with user B on machine M2, each machine must authenticate the other. The machines then form a channel, and agree on a channel key and a verifier. The channel key is used to encrypt all messages from M1 to M2, and the verifier is used to authenticate to M2 users whose secret keys are known by M1. This verification is accomplished by encrypting the verifier with user A's private key. As the encrypted token arrives at M2 it is validated by decrypting it with the public key of user A. These mechanisms are totally transparent to the user. *Authentication Caching* is used to reduce public/private key operations, hence only the first time that a user needs to communicate with a specified remote host is a public/private key operation required.

3. The Authentication Scheme

This section proposes an authentication scheme by stating the goals of the scheme in Section 3.1 and its assumptions in Section 3.2. An overview of the scheme is then presented in Section 3.3.

3.1 Goal of the Authentication Scheme

The goal of our authentication scheme is to authenticate principals for load balancing on remote hosts in a load balancing cluster of machines. In the context of this scheme, authentication means verifying the identity of communication principals to one another. The principals in this scheme are load balancing users.

In order to achieve the intended goal, the valid, potential security threats to the load balancing system must be identified, and counteractive measures must be taken to nullify these threats. The following security threats are considered valid security threats to the system and are counteracted by this scheme:

- The impersonation of an authorized user by assuming the identity of that authorized user. This impersonation can be accomplished easily on a workstation by booting a non-standard kernel and assuming the authorized user's login name.
- The impersonation of an authorized user by replaying the job requests of that authorized user to remote hosts.

- The impersonation of an authorized user by generating that authorized user's correct job request packets. In this situation the job request packets contain either an encrypted authentication string or an encrypted authentication and jobs request string. An unauthorized load balancer user attempts to generate the correct bit sequence (since the unauthorized user does not know the encryption key). To put it simply, the unauthorized user attempts to guess the encryption key.

The third situation mentioned above is distinct from the first two. In this third scenario the unauthorized user attempts to "guess" the encryption key. If the attempt is successful, the unauthorized user can then impersonate the specific user whose unique encryption key has been "guessed", or, in the event that a single global encryption key is used by the system, all authorized users of the system. This impersonation can last only the period of validity of the encryption key. In the first situation the unauthorized user assumes an authorized user's identity. The unauthorized user can then impersonate the authorized user over the duration of the login session, which is usually considerably longer than the period of validity of an encryption key. In the second scenario an unauthorized user captures jobs request packets on the network and replays the packets. This threat is limited in scope in that it is dependent on the remote jobs previously executed by the authorized user. The critical window for this threat is the lifetime of the remote job. The magnitude of this last threat can be dramatically increased if some of the replayed remote jobs grant increased system privileges to the unauthorized user.

The elimination of these security threats comprise the criteria by which this authentication scheme will be proven correct.

3.2 Assumptions of the Scheme

Our authentication scheme assumes the following:

- Remote kernels and users are not trusted until locally authenticated.
- A user only logs on to machines that the user trusts.
- Any machine can inject data into the network.
- Packet smashing (capturing a packet before it reaches its destination, changing the contents, and sending it to its destination) is not possible. Packet smashing is distinct from packet replaying, as in packet smashing the contents of the appropriated packet are altered and replayed, while in packet replaying the contents are replayed without alterations.

3.3 Overview of the Authentication Strategy

In order to achieve the intended goal of the authentication scheme, a viable strategy must be employed. The basis of this strategy is the authentication of the kernels of the machines comprising the load balancing cluster, followed by the authentication, on the desired remote machines, of the load balancing users.

The load balancing cluster consists of a set of hosts, determined by a load balancing configuration file, each executing a unique kernel. The kernel's uniqueness is due to the distinct Authentication Module daemon (AM) that each kernel runs. Each AM is distinct from any other AM by virtue of the unique private/public encryption key pair that it possesses. Each AM also possesses an AM secret key, known only to itself, which it uses for internal security. The internal security of the AM is discussed in Section 4.3. It is important to note that no kernel is able to see the private key or the secret key of the AM of any other kernel.

The authentication of the kernels is achieved by the use of an Authentication Protocol. In the Authentication Protocol, the private/public key pair is used to authenticate the communication principals to each other.

After the authentication of the kernels, the next step in the strategy is the authentication of the load balancing users on the desired remote machines of the cluster. The basis for this user authentication is the restricted knowledge of a secret user token. In this context, restricted knowledge refers to the fact that only the user and the authenticator on the remote host know the secret token. There is an authenticator on each host. The function of the authenticator will be discussed in Section 4.2. Since at any given time only the authenticator and the user know the secret token, it becomes the user's unique identifier. For each machine on which the user wishes to execute, the user must acquire a distinct secret token. As a result, the illegal acquisition of a user's token only allows the impersonation of a user on one specific remote host. The secret token is an encryption key. The authorized user encrypts specified strings in a specified format and sends the encrypted message to the remote authenticator. The strings and formats are known to the authenticator and, upon decrypting the message, the identity of the user is verified. If the decryption of the message results in garbage or the string content or format is incorrect, the user is not an authorized user.

The acquisition of the secret token and the operation of the mechanism by which this token is made known to the remote authenticator and to the user are the responsibility of the AM. The AM communicates with its clients via well-known sockets in order to send and receive remote authentication requests and information.

4. Implementation of the Authentication Scheme

The implementation of the authentication scheme will be presented in four sections. Following the implementation details, an analysis of the implementation will be given. The first section, Section 4.1, gives an overview of the load balancer in which the proposed authentication scheme is to be implemented. The next three sections discuss actions taken at specified times, by the various security processes, to ensure the authentication of users and kernels. Section 4.2 describes the authentication processes invoked at system startup time. The security actions taken at user login time are discussed in Section 4.3. The fourth section, Section 4.4, explains the steps taken to ensure the authentication of the user for each job to be remotely executed. All of the sections mentioned above will be supported by an accompanying diagram. Section 4.5 presents an analysis of the implementation.

4.1 Overview of the Load Balancer

The load balancer in which this authentication scheme is to be implemented is described in detail in [Zhou86]. An overview of this load balancer is now given. This overview will serve as a basis upon which the implementation details of the authentication scheme will be presented. The structure of the load balancing system is shown in Figure 4.1.

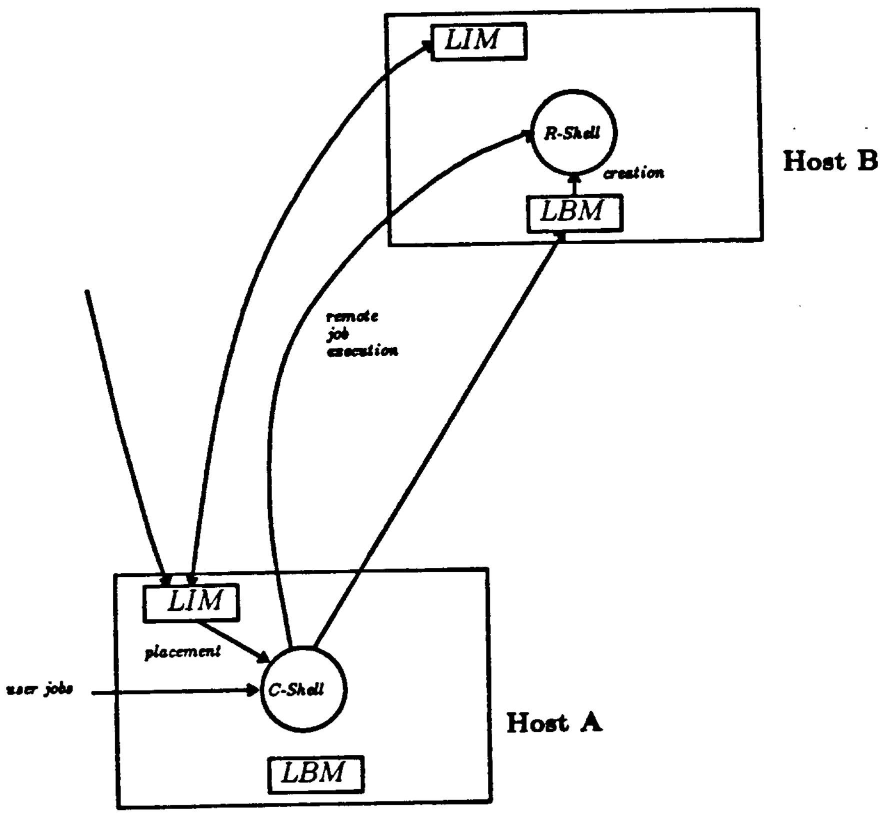

Figure 4.1 Structure of load balancing implementation.

At startup time, the C-Shell reads in a configuration file that specifies the hosts in the load balancing cluster and the names of the jobs that are eligible for remote execution. When an eligible job is submitted by the user to the C-Shell, the C-Shell contacts its Load Information Manager (LIM), a module that constantly monitors the loads of the other hosts in the cluster and performs job placements, for a possible remote host. If a remote host is selected for the job, the C-Shell contacts the Load Balancing Manager (LBM) on that destination host. The C-Shell sends the user name to the LBM, which authenticates the user by determining if the user exists (it checks the 'passwd' structure for the passwd.pw_name and compares it with the username). The LBM then starts up an R-Shell, with the appropriate environment, and establishes a stream connection between the R-Shell and the C-Shell. The R-Shell is kept alive after the execution of the first job so that later commands from the same user login session, to that destination host, do not cause the repeated, costly generation of an R-Shell.

4.2 Authentication Actions at System Startup Time

This section describes the authentication processes invoked at machine boot time. There are three authentication processes invoked at this time. These processes are the load balancing authentication (lba) login process, the AM daemon, and the LBM server. These processes are illustrated in Figure 4.2.

The lba login process is the standard UNIX login process with one added responsibility. This responsibility is to provide the C-Shell with a unique load balancing token. The generation and use of this token is discussed in detail in Section 4.3. It should be noted that the lba login process replaces the standard UNIX login process.

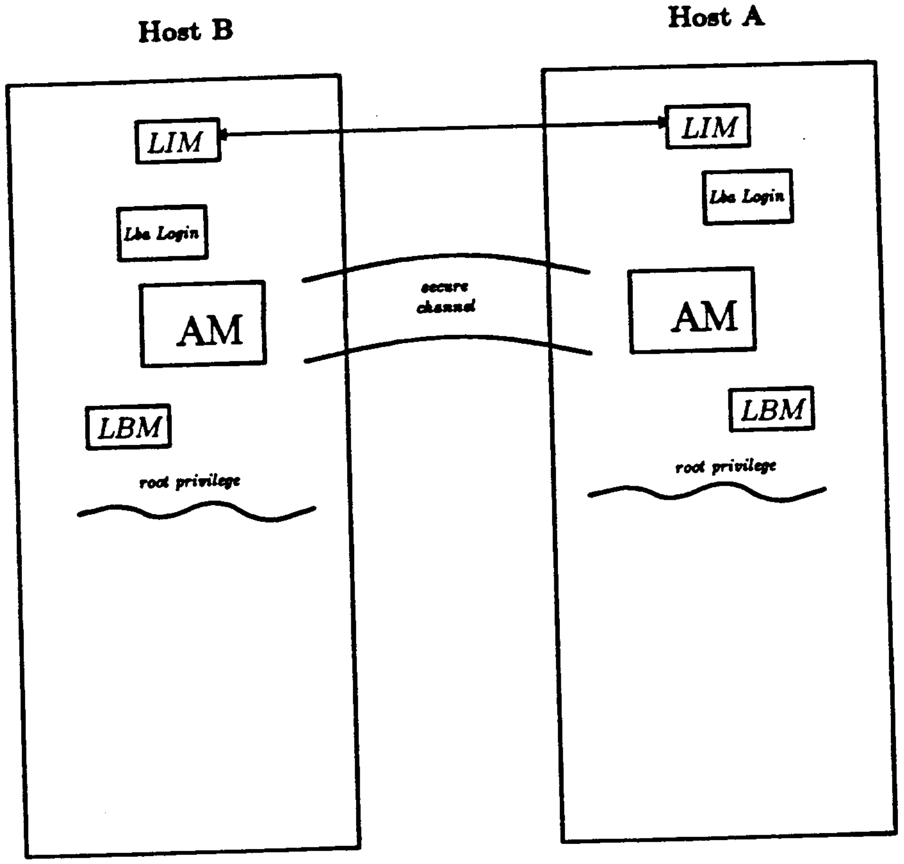

Figure 4.2 Security processes invoked at system startup.

The AM daemon has two functions. The first function of the AM is the generation of the user's unique load balancing token. This token is generated by the local AM and passed to the login process upon request. Further details regarding this token are found in the sections that follow. The second function of the local AM is to support the authentication mechanisms on remote AMs. These mechanisms enable local users to be authenticated on remote hosts. Before the authentication of remote users is possible, remote kernels must be authenticated. A kernel is authenticated by authenticating its AM. Therefore, the AM of the local kernel must authenticate the AM of the remote kernel. Each AM uses its private/public key pair to authenticate a remote AM. This authentication is accomplished by using a private/public key authentication protocol. In this authentication process a secure channel is established by the mutual agreement of a channel or conversation key. In this implementation, the authentication of the AMs and the establishment of a secure channel are accomplished by the use of SUN secure RPC. After the authentication of the AMs and the establishment of the secure channel, the AM on each host can now support the authentication mechanisms on the remote hosts AMs.

The LBM server is the remote user authenticator. This server authenticates the remote user by utilizing information supplied by its local AM. If the user can be authenticated, the LBM creates the appropriate environment in which the remote command is to be executed. If the user cannot be authenticated, the user's request is ignored. The details of this authentication will be presented in Section 4.4. All of these authentication processes operate with root privileges.

4.3 Authentication Actions at User Login

The authentication actions taken upon user login are described in this section. The standard Unix login process accepts as input the username, requests the user password, and upon verification of the password sets up the user environment and "execs" a shell. The lba login process used in this scheme performs one additional action. Upon user login and after login authentication, the lba login process requests and receives from the AM daemon a unique load balancing token for that user. This request and the subsequent reply are made via a well-known socket. This token is passed as a parameter to the C-Shell and is used by the C-Shell to identify itself to the AM daemon. Figure 4.3 illustrates the security action at user login. In Figures 4.3, 4.4, and 4.5, all of the communications between authentication processes have been numbered. These numbers also indicate the order in which the communications take place. In these communications, information is exchanged either by the use of messages, by parameter passing or by interactive input. In the case of messages, the message formats will be shown (along with their numbers in parenthesis), and the format will be discussed. In the case of parameter passing or interactive input, the communication number will be stated and the parameter or input will be discussed. The message format notation is as follows:

- [*first_message*] indicates that the message *first_message* has been sent from one communicator to the other.
- [*field0* : *field1* : *field2*] denotes that a message with three fields, *field0*, *field1*, and *field2* has been sent from one communicator to the other.
- *enc_key*[*second_message*]*enc_key* indicates that the message *second_message* has been encrypted with the encryption key *enc_key* and sent from one communicator to the other.
- [*] indicates an acknowledgement.

Communication 1 is the interactive input of the user's login name and correct password, upon which the login process is initiated. As mentioned previously, the user's unique token is requested during the login process. This request is made by sending a message, via the well-known socket, with the following format:

$$[user] \qquad (2)$$

The AM accepts the username and encrypts it with the AM secret key (amsk). This encrypted string is returned to the login process. The string is encrypted using the DES encryption algorithm [NBS77]. The format of this reply is thus as follows:

$$amsk[user]amsk \qquad (3)$$

The login process then passes this encrypted string (which is the load balancing token) as an argument to the execed C-Shell (communication 4). The AM has a well-known socket to which the login process makes its request. Upon termination of the login process, the user has a unique load balancing token stored in his C-Shell. This token is used as a verifier when the shell communicates with the local AM.

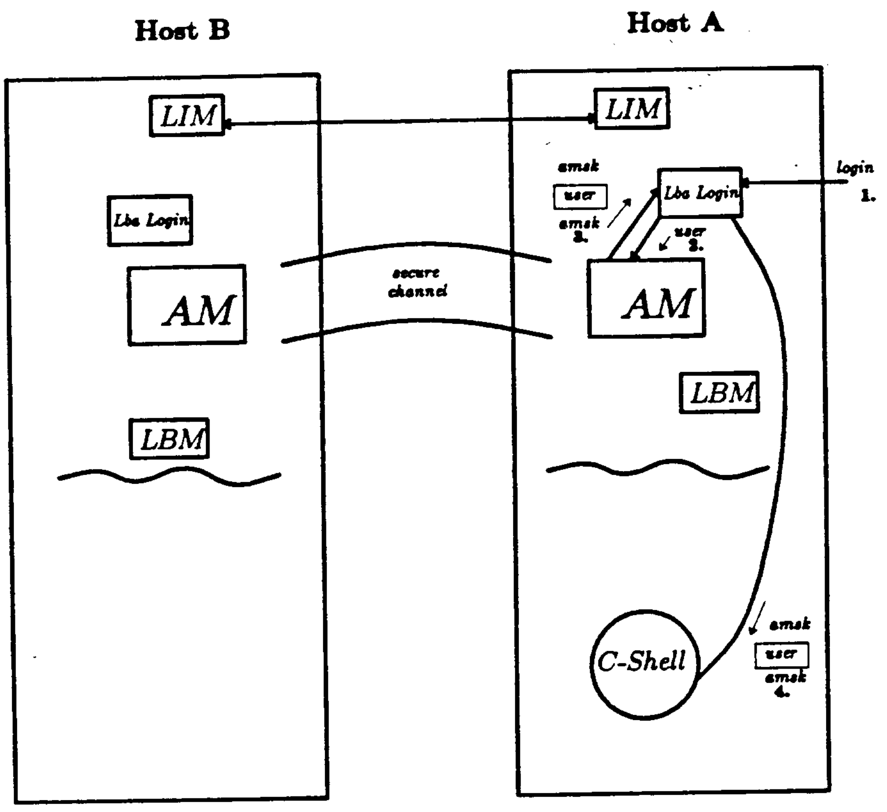

Figure 4.3 Security actions at User Login.

4.4 Authentication Actions at Command Execution

This fourth and final section describes the actions taken to ensure the authentication of the user for each remotely-executable job. Figure 4.4 illustrates these actions and their order.

After querying the LIM about the possible placements of an eligible job, the C-Shell attempts to have the job remotely executed at the host suggested by the LIM. If no previous command from that local host has been executed on the suggested destination host, the user must be authenticated on that host and an R-Shell started up on that host. The C-Shell can then send the command to the R-Shell, where it is executed. If the C-Shell had a command executed on the suggested destination host previously, its R-Shell is present on the host and the C-Shell uses this R-Shell to execute the command. These two scenarios will be discussed with reference to Figure 4.4. In the first scenario, no previous command had been executed at the suggested destination host. Therefore, the user must be authenticated and an R-Shell started up at the destination host. The C-Shell initiates this remote authentication by sending an authentication request to its AM. This request is sent to a well-known socket on which its AM is listening. The request format is as follows:

$$[\mathit{user} : \mathit{amsk}\ [\mathit{user}]\ \mathit{amsk} : \mathit{host}] \qquad (5)$$

The first field contains the username. The second field contains the user's unique load balancing token. As was stated in Section 4.3, each C-Shell has a unique token known only to itself and to its AM. This mechanism ensures that an unauthorized process cannot impersonate a legitimate C-Shell, as no user process has access to the amsk, hence a valid token cannot be generated by that process. The third field contains the name of the suggested remote host.

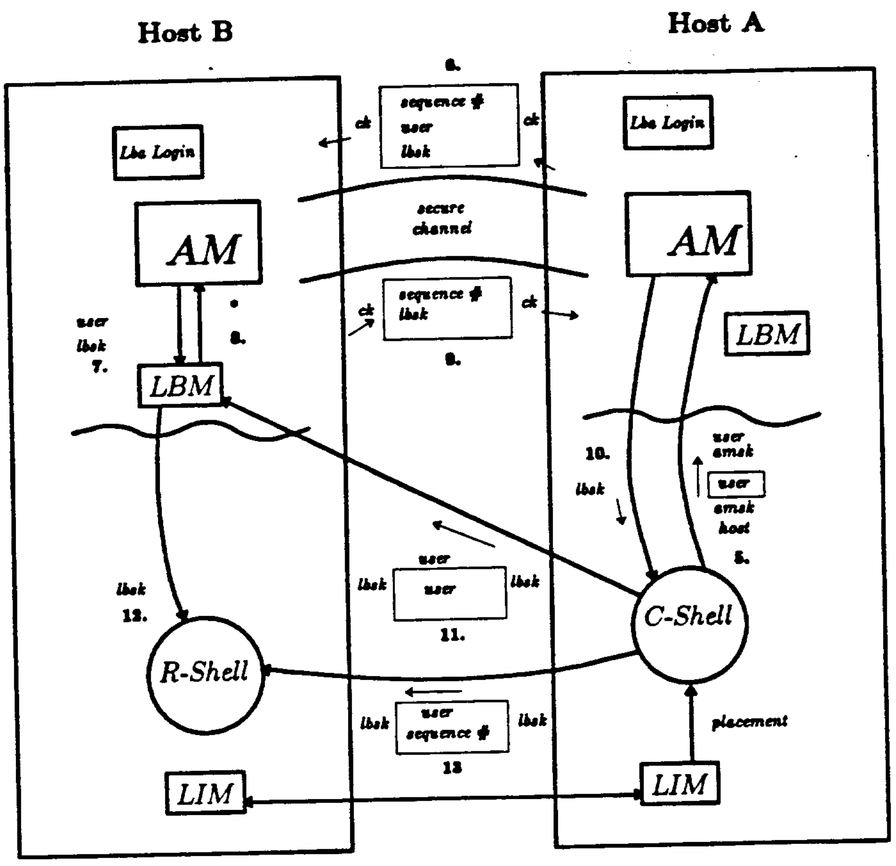

Figure 4.4 Security actions at execution of LB remote command.

Upon receipt of an authentication request, the AM decrypts the token and compares the contents to that of the user field. If they are the same, the user's C-Shell is verified to the AM. The AM then sends an authentication message to the AM of the remote host specified in the request message. The secure channel is used to send this authentication message (message 6). Upon the establishment of the secure channel both AMs were authenticated, assuring that each is part of a legitimate load balancing kernel. This assurance guarantees that the host receiving the authentication message is legitimate. In honoring the C-Shells request, the AM generates a load balancing session key (lbsk) and sends a request to the AM of the remote host, of the form (this message is encrypted in the channel key (ck)):

$$ck[sequence\# : user : lb\_sessionkey]ck \qquad (6)$$

This message is received by the AM on the remote host. The first field of the message contains a sequence number which is used to prevent replays over the secure channel. The second field contains the name of the user who is requesting authentication. Since the local and remote hosts trust each other (this trust was established when the kernels were authenticated), receipt of this message with the correct sequence number indicates to the receiving host that this user is an authorized load balancing user. The last field of message 6 is the lb_session key field. This field contains lbsk, which is the key that will be used by the C-Shell when contacting the remote host's LBM. After the C-Shell is authenticated by the remote LBM, the C-Shell will also use the lbsk when communicating with its R-Shell. The remote AM processes this message by sending message 7 to its LBM.

$$[user : lbsk] \tag{7}$$

The first field contains the username, and the second contains the load balancing session key that will be used by that user. The LBM stores the username and the lbsk until they are needed, and returns a simple acknowledgement to the AM. This acknowledgement is shown in message 8.

$$[*] \tag{8}$$

The remote AM then sends a message to the requesting AM. The format of this message is :

$$ck[sequence\# : lbsk]ck \tag{9}$$

The sequence number field prevents replays, while the lbsk field informs the local AM that all of the authentication mechanisms have been set up on the destination host. The local AM then passes the lbsk to its C-Shell (communication 10). This indicates to the C-Shell that all authentication mechanisms have been set up on the remote host.

At this point, the C-Shell establishes a stream connection with the LBM on the destination host and sends an R-Shell startup request to that LBM. This remote LBM then attempts to authenticate the C-Shell. The C-Shell request format is shown below:

$$[user : lbsk[user]lbsk] \tag{11}$$

The first field identifies the user. The next field contains the user verifier. This verifier consists of the username encrypted in the lbsk. This second field is used to authenticate the user. The LBM uses the first field, username, to query the database for that username and retrieve the corresponding load balancing session key (lbsk). The LBM then uses the lbsk to decrypt the second field of the message. If the decrypted second field is the same as the username, the user is authenticated and the LBM starts up an R-Shell for that user. Upon R-Shell startup, the authenticated user's username and lbsk are flushed from the database. This flushing ensures that a replay of a previous R-Shell request is ignored. If a previous request is replayed, the LBM will have no entry for that user in its database (the entry would have been flushed previously) and therefore the request will be denied. If the decrypted field does not match the username or the username is not present in the database, a denial is sent to the requesting C-Shell.

Upon startup of an R-Shell, the LBM passes the lbsk as an argument to that R-Shell. The passing of the lbsk is illustrated as message 12. Successful transmission of messages 5 through 12 results in the authentication of a user on a remote machine and the startup of an R-Shell on that machine. It should be noted that R-Shells started on the remote host are on a per C-Shell basis, and therefore the R-Shells accept jobs only from their C-Shell. This one-to-one correspondence is achieved by the use of the unique lbsk and sequence#. A user with multiple load balancing C-Shells on the same machine will have R-Shells on destination hosts corresponding to each individual C-Shell. The C-Shell can now use the format of message 13 to pass remotely executable jobs to the R-Shell.

$$lbsk[user : sequence\#]lbsk \tag{13}$$

Message 13 is placed at the beginning of each remote job sent to the R-Shell. This message allows the R-Shell to verify the authenticity of the job. The R-Shell decrypts the message with its lbsk. If the decrypted message is not in the format of message 13, the job is discarded. If the decrypted message is in the correct format, the R-Shell verifies that the user field contains its username, and that the sequence# field contains the correct sequence#. Sequence#'s begin from 0 and are incremented by 1 for each consecutive job. If the username field does not contain the R-Shell's username or the sequence# is incorrect, the job request is ignored. An incorrect sequence# is one which is less than or equal to the previous sequence# (this situation could be caused by a replayed message) or one which exceeds the previous sequence# by 2 or more (this situation could be caused by the attempted generation of the correct job packet by an unauthorized user listenning on the network).

The other scenario to be considered is one in which previous commands have been sent to the suggested remote host. This scenario necessitates that an R-Shell had been previously started on this host. In this scenario, the C-Shell checks its internal database for the lbsk and sequence# pertaining to that host, and corresponds directly with the R-Shell using the format of message 13. Messages 5 through 12 are sent once per user per machine. The entire authentication scenario is illustrated in Figure 4.5.

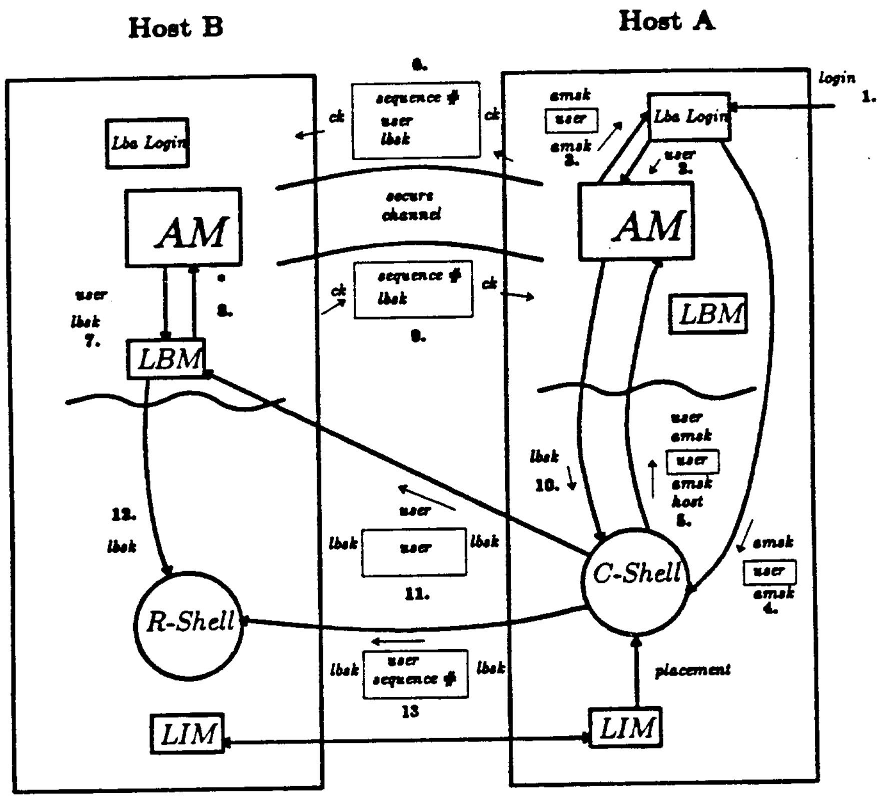

Figure 4.5 Authentication Scheme.

4.5. Implementation Analysis

This section is comprised of three subsections. The first subsection, Section 4.5.1, discusses the countermeasures taken to satisfy the security criterion stated in Section 3.1. A discussion of the techniques employed by the implementation in the scenario of a remote host failure is given in Section 4.5.2. This section is concluded by a discussion of a pertinent implementation issue in Section 4.5.3.

4.5.1. Analysis of Countermeasures

This section analyzes the security threats and subsequent countermeasures taken by the implementation. The first threat addressed was the impersonation of a user by booting a non-standard kernel and using an 'su' command. In our scheme, a non-standard kernel would not possess a correct AM (a correct AM is one which possesses the unique AM private and secret keys) and therefore will never be authenticated by other AMs. Hence, no user on that machine will be allowed to participate in load balancing.

The next threat addressed was the impersonation of another user by replaying job requests to remote hosts. There are three connections on which the replaying of messages pose a security threat. The messages sent along these connections are, respectively, messages 6, 11 and 13 (see Figure 4.4). The threat of replay of messages 6 and 13 is counteracted by the use of sequence numbers. The details of this countermeasure were discussed in Sections 4.4. The threat of a replay of message 11 is counteracted by the flushing of the appropriate information from the LBM database after the startup of the R-Shell. The details of this flushing were presented in Section 4.4.

The third and final threat addressed was the impersonation of another user by generating correct job request packets. Connections supporting messages 11 and 13, respectively, are vulnerable to this threat. The use of an lbsk which is random, short-lived, and distinct is the measure used to counteract this threat. Because the lbsk is random, it is difficult for a user to generate the correct bit patterns. Due to the short-lived nature of the lbsk (the lbsk life span is one user login session), the analysis of the job request bit patterns by an unauthorized user will reveal negligible information and therefore cannot be used to pose a significant security threat. The distinct nature of the lbsk lies in the fact that each R-Shell/C-Shell pair in the load balancing system shares a unique lbsk. This nature makes it difficult for an unauthorized user to determine the encrypted job request headers, as the same job sent to different R-Shells will have different headers due to the unique encryption key.

4.5.2 Analysis of Crash Recovery Techniques

In this section the robustness of the authentication scheme is analyzed in the situation where a remote load balancing host crashes.

Before proceeding with the analysis, three terms must be defined. These terms are robust, authentication process, and authentication process state. An authentication scheme is said to be robust if the failures of one or more remote hosts in the load balancing cluster do not affect the validity of the scheme on a surviving local host and if, upon recovery of the previously failed hosts, their authentication mechanisms are not compromised. An authentication process is the sequence of messages, discussed for our scheme in Sections 4.2, 4.3, 4.4, which are required to authenticate a remote user, set up the R-Shell, and guarantee secure handling of job requests to the R-Shell. An authentication process exists between two machines and is said to have completed when messages 1 to 12 have been successfully transmitted and at least one instance of message 13 has been transmitted. This process has three process states. Each state is defined by the messages that have been previously transmitted. An authentication process is in state 0 when the C-Shell has requested authentication on a remote host and has received a lbsk. The boundaries of this state are messages 1 to 10. State 1 is bounded by messages 11 and 12. After completion of state 1 the local C-Shell has been authenticated by the remote LBM , an R-Shell has been set up, and the C-Shell has been connected to the R-Shell. State 2 is that state at which the C-Shell has an R-Shell on a remote host and need send only instances of message 13 in order to execute jobs.

To ensure that the authentication scheme remains robust during and after single or multiple remote host crashes, it is essential that the local authentication process handles a remote host failure without any breech in security, regardless of the state of the local authentication process at the time of failure. Therefore, to prove that this scheme is robust, the handling of a remote host failure at each authentication state must be discussed.

If the local authentication process is at state 0, the messages of importance are messages 6 and 9. With a remote crash occurring before message 6, there is no security violation as the local AM cannot send a message to the remote AM, since the secure channel no longer exists. If the AM tries to use this channel, it is immediately made aware of the crash. After retries, the AM declares the host "down" and discards that host's ck. The local AM then sends a message to the C-Shell indicating that load balancing on that host is not possible. The C-Shell now has the option to execute the job locally or choose another host. If a crash occurs on the remote machine after message 9 is sent to the local machine, the C-Shell receives the lbsk from the AM. Although the lbsk is invalid, this does not cause a security violation. In this state, state 0, there are no security violations caused by a remote crash.

If the authentication process is in state 1 and a remote crash occurs, the stream socket along which the C-Shell and LBM communicate will be disconnected, thereby indicating to the C-Shell that a crash occurred. Subsequent retries will confirm the C-Shell's hypothesis. The C-Shell then discards the lbsk and can either execute the job locally or try another host. State 1 does not foster any security violations.

If the authentication process is in state 2 and a remote crash occurs, the stream socket connecting the R-Shell with the C-Shell is disconnected. This indicates to the C-Shell that a crash has occurred and, after unsuccessful retries, the lbsk is discarded and the job is either done locally or sent to another remote host. There are no security violations in state 2.

Thus, in all of the possible authentication process states, a remote crash does not cause any violation in the security of the scheme. It should be noted that the remote host does not maintain any record of previous authentication transactions. While the lack of transactions records delays the recovery time of the authentication mechanism, it also prevents any security problems that may arise from the crash. A typical problem that may arise is the availability of user lbsks in the core dumps of the crashed kernel. This situation, however, is not a problem as the recovered host discards all lbsks, cks, and data previously used by the authentication scheme. As mentioned previously, all lbsks and the ck common to the crashed remote host are discarded by the local host. The secure channel must then be re-established and all previous load balancing users must be authenticated again. Although the authentication recovery actions are costly, this strategy prevents security violations on either side of the authentication process, and the authentication partition/merge problems that would be encountered in a distributed environment are avoided.

The analysis above clearly indicates that the implemented authentication scheme is robust.

4.5.3 Implementation Issues

The issue addressed in this section is that of the secure storage of the AM private key and of the AM secret key in order that no user may, upon inspection of the kernel code, determine these keys. These AM keys will be encrypted in an AM password (this password may be the same as the superuser password) and stored in a file on the file server. This encryption will prevent any other user or kernel from accessing the keys of another AM and impersonating that AM. When an AM is started, the AM password will be supplied by the superuser and used to decrypt the AM private and secret keys. It should be noted that public keys are either stored in a secure database in each host, or stored in a nameserver and made available through authenticated request. This secure storage will prevent an unauthorized user from broadcasting his public key and becoming a member of the load balancing cluster.

5. Experiments

This section describes the experiments conducted on our authenticated load balancer and provides an analysis of their results. This description and the subsequent analysis are presented in four subsections. The first of these subsections, Section 5.1, discusses the motivations for and the goal of the experiments. The design of the experiments is then examined in Section 5.2. The next subsection, Section 5.3, gives a description of the experimental environment. The results of the experiments and an analysis of these results are provided in Section 5.4.

5.1 Goal of Experiments

As stated in Section 1, the security concerns, in particular that of authentication, must be addressed in order to foster widespread use of load balancing. Not only must authentication be present, but it must not substantially diminish the performance gains provided by load balancing. It is with this motivation that our experiments were devised. The purpose of the experiments is to determine the degradation of the performance gains due to the use of authentication in a load balancing system.

Performance gains can be measured by many indices, such as throughput rates, response times, and resource utilizations. Since the environment of interest is an interactive computing environment, a performance index is needed that accurately assesses the effect of our authentication scheme on the load balancing system. Therefore, the primary performance index that was chosen was the mean job command response time. Another important index, the standard deviation of response times, was also measured in our experiments.

The goal of the experiments is to determine the effect of authentication on the mean job response time of all jobs executed during a measurement session in the load balancing environment under analysis. This environment has been described in Sections 1 and 4.1.

5.2 Design of Experiments

This subsection examines the basis for the design of the experiments.

As discussed in the previous section, the main performance index that was chosen was the mean job response time of all jobs executed during a measurement session. It was difficult, however, to determine the response times of jobs executing remotely in the background, due to the implementation of the load balancer. In the load balancer's implementation, some of the jobs executing remotely in the background (for example, *cc*, *ditroff*, or *grn*) do not acknowledge their completion to their local hosts, hence their response times cannot be determined. To circumvent this difficulty, we made use of the system accounting facility to obtain the response time of each process executed during a measurement session. In most cases the two indices, job response time and process response time, are the same, as only one process is created per job. However, for a few commands (for example, *ditroff, grn, lint, cc*) several processes are created. In these cases, the response time of each created process is considered when computing the mean response time. The overhead of load balancing and authentication (if present) is measured during the session by the load balancer. This overhead is then added to the process response times and the mean process response time is calculated. The mean process response time calculated in this way is the performance index that is used.

With the method of calculating the performance index resolved, the next step was to determine the experiments to be conducted, thereby ensuring that meaningful results would be obtained. To fully assess the effects of authentication on the load balancing system, the following data were required:

- The mean job response time for all jobs in a measurement session with both load balancing and authentication disabled.
- The mean job response time for all jobs in a measurement session with load balancing enabled and authentication disabled.
- The mean job response time for all jobs in a measurement session with both load balancing and authentication enabled.

Three sets of measurements are needed to obtain these data. These data allow the evaluation of the impact of authentication on the mean job response time of the load balancing system, using the mean job response time of the system without load balancing as a basis for comparison. The data are calculated from measurements of the "typical scenario" response time. The "typical scenario" is the scenario in which the user's C-Shell has been previously authenticated, an R-Shell has been started on the destination host, and both the C-Shell and the R-Shell possess the lbsk. (This scenario has been fully discussed in Section 4.4.) At this point, the C-Shell sends an authenticated job request to its R-Shell, which verifies the job request and executes any request whose authenticity has been verified. In obtaining the data mentioned above, only "typical scenario" measurements are taken, since this produces a typical estimate of the response times.

To complete our analysis, the overhead of initializing the authentication mechanism and performing the authentication, on a per user basis, is also measured. This measurement includes the LBM's query to its authentication database. Details of this authentication process were given in Section 4.4.

5.3 Experiment Setup

The environment of interest is an interactive, distributed workstation environment in which the workload consists of a mixture of remotely executable jobs and locally executable jobs. The remotely executable jobs that are of importance are discussed later in this section. In order to set up the appropriate experimental environment, specific decisions were made with regard to the hardware, the workload, and the placement algorithm. Each of these considerations is discussed below.

The hardware environment in which the experiment was conducted consisted of a group of 3 SUN 3/50's and 1 SUN 2/100 connected by a 3Com Ethernet. Each of the SUNs used was diskless and all were supported by a single file server. This configuration provided us with an interactive computing environment commonly used to support software development. Measurements were usually taken during the "graveyard shift" (between the hours of 12:00 am and 6:00 am) to avoid deviations in the data due to the presence of other users on the network.

The workloads that were used in the experiments were modifications of those employed by Zhou and Ferrari in their experiments [Zhou86]. The basis of Zhou and Ferrari's workload was a trace of a production VAX-11/780 running under the Berkeley Unix 4.3 BSD system [McKusick85] for an extended period of several months. This trace was analyzed, and a number of frequently executed commands were selected and used to construct the scripts which are used to generate the workload.

command	elig.	function	command	elig.	function
cat	*N*	*view a file*	*ls*	*N*	*directory list*
cc	*Y*	*C compiler*	*man*	*Y*	*man page view*
cp	*N*	*file copying*	*mv*	*N*	*move file*
date	*N*	*current time*	*nroff*	*Y*	*text formatter*
df	*N*	*file sys. usage*	*ps*	*N*	*process chech*
ditroff	*Y*	*formatter*	*pwd*	*N*	*current dir.*
du	*N*	*disk usage*	*rm*	*N*	*delete file*
egrep	*Y*	*text pat. search*	*sort*	*N*	*file sorting*
eqn	*Y*	*enq. formatter*	*spell*	*Y*	*spelling check*
fgrep	*Y*	*text pat. search*	*tbl*	*Y*	*table formatter*
finger	*N*	*user info.*	*troff*	*Y*	*text formatter*
grep	*Y*	*text pat. search*	*uptime*	*N*	*system uptime*
gren	*Y*	*graph plotting*	*users*	*N*	*list of users*
lint	*Y*	*C prog. checker*	*wc*	*N*	*word count*
lpq	*N*	*Printer queue*	*who*	*N*	*user info.*

Table 1. Load Balancing Eligible Commands

Table 1 gives a list of the commands used in the scripts and their eligibility for remote execution. To obtain various intensities of load, typical of multi-user systems, we ran a number of jobs in the background. User think times were simulated by the "sleep" command. The scripts that were used in the experiments produced a moderate load (between 60 % and 70 % of CPU utilization) on the system, and each host ran a different script in order to avoid the problem of synchronization. Synchronization may occur when the same script is executed on each host and the scripts are started at the same time. In this situation, each host simultaneously executes the same job; therefore, the effectiveness of load balancing is at a minimum, as the average and even the instantaneous load of each of the hosts tend to be very close to those of the other hosts in the cluster. The duration of each script was approximately 45 minutes. The same experiment was repeated a number of times (usually 6), and the mean and standard deviation of the indices were computed over these replications. In determining if the number of replications was sufficient to obtain meaningful results, we calculated the ratio of the standard deviation of the 6 response time means around their mean to the mean of the 6 response times for each measurement. The ratios (which are the values of the coefficient of variation for the 6 samples we had in each case) in the no load balancing case, the load balancing without authentication case, and the load balancing with authentication case were 13.2 %, 9.72 %, and 6.6 %, respectively. These results indicate that the use of 6 replications was sufficient to obtain meaningful data.

The placement algorithm used was the Global algorithm [Zhou86]. This placement algorithm was chosen because thorough measurements have been performed with this algorithm [Zhou86]; thus the validity of our measurements could be better gauged.

5.4 Results and Analysis

The results of the experiments are shown in Table 2.

Cases	Resp. Time	Increase	Std. Dev.	Increase
NoLb	15.90	0 %	33.18	0 %
Lb	13.40	15.72 %	25.10	24.35 %
AuLB	14.51	8.1 %	30.24	8.81 %

Table 2. Results of Experiments

Table 2 displays the mean job response time and the standard deviation for each of the three cases under analysis. These three cases are : no load balancing, load balancing without authentication, and load balancing with authentication. In this table, the percentage decreases in response times and standard deviations with respect to the no load balancing case are also presented. The response times shown in Table 2 are "typical scenario" times. As mentioned previously, in a "typical scenario" the user's C-Shell has been remotely authenticated. The time taken to initialize the authentication mechanism and to authenticate the user's C-Shell on each desired remote machine is approximately 250 ms. This authentication is done once per user per login session per desired remote host.

As stated in the previous section, the mean job response time is used as a measure of performance. Specifically, a performance gain is realized when the mean job response time is decreased. Using this performance measure, it is evident from the results shown in Table 2 that there is a performance gain in the system even when authentication is present in the load balancing system. The improvement in performance, however, is not as great as that realized when load balancing is implemented without authentication, which is to be expected. The standard deviations, which measure the predictability of the system, also display similar results. As shown in Table 2, the mean job response time is decreased by 7.62 % when authenticated load balancing is implemented. Thus, the performance gain achieved by the load balancer when no authentication is present is decreased by 48.47 %.

If we consider the question about how this degradation would vary when the characteristics of the load change, we can argue that at the two extremes of the spectrum of such changes the results presented above can be extrapolated in two ways. One conjecture is that the decrease in performance gains due to authentication (7.62%) is constant and is independent of the intensity of the workload. The second conjecture is that the decrease in performance due to authentication (48.47 % of the original gain) is a linear function of the workload intensity. We will now examine these two conjectures and give an assessment of their respective merits.

Since both conjectures attempt to predict the effect of authentication at different load intensities, it is necessary that we characterize load intensities in our environment before proceeding with the examination. The intensity of the workload on a system is dependent on the size of the active jobs and the number of active jobs present on that system. In our interactive development environment, we believe that the statistical distribution of active job sizes is approximately constant throughout the system. The intensity of the workload in our environment is almost entirely dependent on the arrival rate of active jobs. An increased job arrival rate is usually due to an increase in the number of users on the system.

In the first conjecture, the decrease in performance gains is constant and independent of the intensity of the workload. Since the intensity of our workloads is dependent on the number of active jobs present in the system, this conjecture implies that the decrease in performance gains is independent of the number of jobs in the system. This implication is questionable as most of the authentication overhead in the "typical scenario" is as a result of the encryption and decryption operations performed, on a per remotely executed job basis, to support the authentication mechanism. (A description of the encryption and decryption operations has been given in Section 4.3.) Statistically, an increase or decrease in the number of active jobs in the system results in an increase or decrease in the number of jobs eligible for load balancing. (Note that, the intensity of the workload does not affect the ratio of eligible to ineligible jobs.) For this reasons, we conclude that it is unlikely that the decrease in performance gains is constant over all workload intensities, as the number of eligible jobs will increase or decrease with the intensity of the workload.

The second conjecture states that the decrease in performance gains is linearly dependent on the workload intensity. This implies that the decrease in performance gains is linearly dependent on the number of jobs in the system. This is a plausible explanation, as will be shown by the following argument. When the system is heavily loaded, the increase in the number of active jobs causes an increase in total authentication overhead, thereby reducing the performance gains due to load balancing. However, this increased load intensity also causes an increase in the performance gains due to load balancing [Zhou86]. When the system is lightly loaded, the decrease in the number of active jobs reduces the authentication overhead. This reduction in overhead causes a reduction in the decrease of the performance gain due to load balancing. But, as shown in [Zhou86], at low load intensities the performance gain due to load balancing is at its minimum. In both of the cases mentioned above, the authentication mechanism imposes a performance reduction that is approximately linearly dependent on the number of jobs, while the load balancer provides a gain increase that increases with the number of jobs [Zhou86]. This suggests that the two linear factors, the augmenting factor provided by load balancing and the reduction factor due to authentication, can be combined to give a linear relationship that is consistent over a wide range of workload intensities.

In assessing the conjectures presented above, it is obvious that the first cannot be supported, while the second can be supported. It should be noted that this assessment is not intended to be thorough, but rather to serve as a prompt for further thought and experimentation on the effects of authentication in load balancing under various workloads intensities.

6. Conclusion

In this paper, we proposed an authentication scheme for use in a load balancing environment, implemented this scheme, and utilized experiments to evaluate the performance of our implementation.

Our authentication scheme was designed to counteract three specific types of impersonation threats. The counteraction of these security threats can be achieved by the authentication of both kernels and users. The kernels are authenticated using a public/private key authentication protocol, while the users are authenticated by using DES encryption with a shared conversation key.

In describing our implementation, we discussed the sequence of actions taken at each interval during remote authentication. An analysis of the implementation issues pertinent to our distributed workstation environment was then presented.

The mean response time was used as the primary index to assess the performance of the authentication scheme. Measurement results showed that there was a performance gain over that of an unbalanced system, and that the performance gain due to load balancing without authentication was reduced by the introduction of authentication, as we expected. A conjectural assessment of the effect of different workload intensities was then presented. This assessment was meant to stimulate thought and further experimentation on the many unaddressed issues raised by the design and implementation of an authenticated load balancer.

It is hoped that this work will lay the cornerstone upon which the newly accumulated knowledge on authenticated load balancers will be built.

Acknowledgements

My eternal gratitude is first extended to Professor Domenico Ferrari without whose experience, support, and perseverence this work would not have been. I also proffer deep thanks to Stuart Sechrest. Stuart's expertise in network issues buffered me against many wasted days. His patience and useful suggestions will always be remembered. My thanks also go to Harry Rubin, Venkat Rangan, and Songnian Zhou for their time and effort spent in guiding this work.

References

[Anderson87]

Anderson, D., Venkat Rangan, P. 1987. A Basis for Secure Communication in Large Distributed Systems. *University of California, Berkeley/Computer Science Department Technical Report.* January 1987.

[Barak84]

Barak, A., Shiloh, A. 1984. A Distributed Load Balancing Policy for a Multicomputer. *The Hebrew University of Jerusalem/Department of Computer Science Technical Report.* 1984

[Bershad85]

Bershad B. 1985. Load Balancing with Maitre d'. *University of California, Berkeley/Computer Science Department Technical Report.* December 1985.

[Birrel82]

Birrel, A., Levin, R., Needham, R., Schroeder, M. 1982. Grapevine: An exercise in distributed computing. *ACM Communications.* 25(4):260-274.

[Birrel85]

Birrel, A. 1985. Secure Communication Using Remote Procedure Calls. *ACM Transactions on Computer Systems.* 3(1):1-14.

[Ezzat86]

Ezzat, A. 1986. Load Balancing in NEST: A Network of Workstations. *Proc. 1986 Fall Joint Computing Conference Dallas, TX* 4():684-690.

[Hagmann86]

Hagmann, R.1986. Process Server: Sharing Processing Power in a Workstation Environment. *Proc. 6th International Conf. on Distributed Computing Systems Cambridge, MA.* 5():260-267.

[Hwang82]

Hwang, K., Croft, W., Goble, G., Wah, B., Briggs, F., Simmons, W., Coates, C. 1982. A UNIX-based Local Computer Network with Load Balancing. *IEEE Computer* 15(4):55-66.

[Joy83]

Joy, W., Cooper, E., Fabry, R., Leffer, S., McKusick, K., Mosher, D. 1983. 4.2BSD System Manual. *University of California, Berkeley/Computer Systems Research Group Report.* July 1983.

[Linvy82]

Linvy, M., Melman, M. 1982. Load Balancing in Homogeneous Broadcast Distributed Systems. *Proc. ACM Computer Network Performance Symposium* 4():47-55.

[NBS77]
NBS 1977. Data Encryption Standard. *FIPS publication 46, NBS, U.S. Dept. of Commerce Washington* 1977

[McKusick85]
McKusick, K., Karels, M., and Leffler, S. 1985. Performance Improvements and Functional Enhancements in 4.3 BSD. *Summer Usenix.* June 1985:519-531.

[Taylor86]
Taylor, B., Goldberg, D. 1986. Secure Networking in the Sun Environment. *Summer Usenix.* 1986:27-36.

[Zhou86]
Zhou, S., Ferrari, D. 1986. An Experimental Study of Load Balancing Performance. *University of California, Berkeley/Computer Science Department Technical Report.* July 1986.