Timely Authentication in Distributed Systems

Kwok-Yan Lam* and Thomas Beth

European Institute for System Security, Karlsruhe, Germany

Abstract. This paper discusses the use of time in distributed authentication. Our first objective is to give reasons for the provision of authentication protocols whose correctness depends on the correct generation of timestamps. Our second objective is to explain that this proposal is not, at least theoretically, as insecure as it first seems to be. The conclusion of this paper motivated our current effort of designing a secure clock synchronization protocol as a part of our overall goal of building a secure distributed system.

Key words Distributed Operating Systems, Authentication Protocols, Interprocess Communications

1 Introduction

In a traditional single machine environment, computer security can be effectively enforced by the operating system. High levels of security can be achieved purely through careful construction of the operating system software. The hardware allows an operating system to run processes in such a way that they cannot interfere with each other and with the operating system itself.

In a general distributed environment, because of the autonomy of the machines and the open nature of the communication network, security can no longer be provided and enforced by operating systems alone. Therefore, in an open system, all machines on the network have to make their own arrangements to ensure the privacy, integrity and timeliness of each message received from the network.

Distributed authentication, whose primary goal is to prevent impersonation, is a common way to approach this problem. It is a fundamental mechanism for the provision of security in computer systems in that authorization (identity based access control), accounting and resource management mechanisms could be circumvented if masquerading attempts were possible.

Authentication protocols are procedures by which various pairs of communicants (we call them principals [2]) satisfy themselves mutually about each other's identity. Often, by the end of a successful run of the protocol, if the principals

* Usual address of author: Computer Science Department, Royal Holloway, University of London, U.K.

[2] A principal is an entity whose identity can be authenticated, according to the international standard ISO/IEC 9798-1 [5]

Y. Deswarte et al. (eds.), *Computer Security - ESORICS 92*

involved are really who they say they are, then they will end up in possession of one or more shared secrets for use as encryption keys to secure interprocess communications. For this reason, some authentication protocols are also called key distribution protocols.

Another goal that authentication protocols aim to achieve is freshness of messages involved in any run of the protocols. The requirement of freshness ensures that replays of messages previously tapped from the network do not compromise security of the protocol. Any principal who may be confused by illegitimate replays of old messages needs to find a way to ensure that only fresh messages are accepted for processing. There are a variety of ways to achieve freshness including

- the use of challenge-response operations: a principal A expecting a fresh message from another principal B first sends a nonce identifier (challenge) to B, and requires that the subsequent message (response) received from B contains the correct nonce value. In this case, it is the responsibility of A to choose a sensible value for the nonce identifier;
- the use of time: a principal A accepts a message as fresh only if the message contains a timestamp which, in A's judgement, is close enough to A's knowledge of current time. This requires, in addition to the existence of a globally accessible clock, that communicating principals are reasonable and do not generate postdated messages unnecessarily.

Most existing distributed systems employ one or the other of these approaches in their authentication mechanisms to enforce security. This results in two main classes of authentication protocol: challenge/response-based authentication and timestamp-based authentication.

One aim of this paper is to discuss how authentication mechanisms should be provided in order to meet the communication requirements of most general purpose distributed systems. A close inspection of the interprocess communication primitives most commonly used in distributed systems suggested that challenge/response-based authentication can be satisfactorily incorporated into connection-oriented communications while timestamp-based protocols are more appropriate for connectionless interactions. Hence, we suggest that the provision of both classes of authentication mechanisms is desirable.

However, regarding the use of timestamp-based protocols, the philosophy of designing a security protocol whose strength relies on a vulnerable one, the clock synchronization protocol, is doubtful. In this paper, we also explain that the provision of timestamp-based authentication is not, at least theoretically, as insecure as it first seems to be.

We describe these two classes of authentication protocols in the next section. We then give a discussion of the features of the ways that processes in a distributed system (without shared memory) communicate. This suggests that authentication protocols of both classes should be provided in a system in order to support secure communications for different applications. Finally, we discuss

the security of timestamp-based authentication protocols. Our conclusion of this discussion regarding how authentication should be provided is different from the views of other researchers.

2 Two Classes of Authentication Approach

A number of strong authentication protocols have been developed in the past, some based on the use of conventional shared-key cryptography while others are based on the use of public-key cryptography. Using the X.509 [18] terminology, a strong authentication mechanism is an authentication mechanism which relies on the use of cryptographic techniques to protect the exchange of protocol messages[3].

One or more trustworthy third parties, called authentication servers in shared-key cryptosystems or certification authorities in public key cryptosystems, are usually needed. An authentication server shares a key with each principal in the system and typically generates new session keys for communication between the principals. A certification authority, which may be an off-line entity, provides certificates for principals' public keys. Public key certificates provide a trusted binding of a principal's name to a public key.

Both authentication servers and certification authorities are trusted to make proper use of the data they hold when executing authentication protocols. Authentication servers are often trusted to generate new secrets in a proper manner. A certification authority which has a well-known public key is trusted to pass on the public keys of the principals. We use the term Key Distribution Center (KDC) to refer to the trusted third party.

With respect to the ways of guarding against message replays, authentication protocols can roughly be classified into challenge/response-based authentication and timestamp-based authentication. Challenge/response-based protocols require one or more steps of handshake between the KDC and a pair of communicating principals wanting to establish mutual authentication. The pair of communicating principals is usually called a client-server pair. The client principal initiates the communication session while the server principal awaits messages from the initiating principal. Using the ISO/IEC 9798-1 [5] terminology, the client principal and the server principal in our discussion are known as the authentication initiator and the authentication responder respectively.

Examples of this class include [4], [11] and [13]. These protocols assure both client and server of the freshness of the interaction by means of nonce identifiers and handshake. A general principle about the use of nonce identifiers is that the identifier should always be generated by the party that sought reassurance about the time integrity of a transaction [12].

A shared secret between the communicating principals is usually established

[3] Whereas, simple authentication relies on the originator simply supplying its name and password which are checked by the recipient.

at the successful completion of the protocols. The principals may use this shared secret as an encryption key to protect their communications. Often sequence numbers are used to ensure the freshness of messages to be exchanged thereafter. A secure communication channel may therefore be established between them.

One of the disadvantages of using challenge/response mechanism is that, because of the use of multi-step handshake, it rules out the possibility of authenticated datagrams. More seriously, all servers must then retain state information, the nonce identifier, to complete the authentication protocol. In a distributed system that employs the client/server model of system structuring, this requirement of maintaining state information by the server degrades scalability of the system, and hence should be avoided whenever possible.

Timestamp-based protocols, also with the participation of the KDC, allow mutual authentication by means of a pair of protocol messages, one in each direction, between the communicating principals. Timestamp-based authentication begins with the client requesting a timestamped ticket from the KDC for communication with the server. When the client wants to communicate with the server, in addition to the message proper, it sends this ticket to the server as a proof of the claimed identity.

Protocols of this class include [3], [9], [18] and [17]. Timeliness of these protocols is assured by the incorporation of timestamps in each encrypted message. If a shared-key cryptosystem is used, the ticket usually carries one or more secrets known only to each legal participant of the authentication protocol. The shared secrets are made known to the communicating principals at the end of the protocol, and knowledge of them implies authenticity. It is also possible to protect the communication channel using the shared secrets.

A timestamp-based protocol has the advantage that, within the lifetime of a valid ticket, each message can be self-authenticated and that delays before secure communications start can be reduced. This feature is especially desirable for distributed applications which use connectionless communication service, e.g. Remote Procedure Call and UDP/IP, as a major mechanism for interprocess communications.

In addition, with the application of timestamp-based authentication in a client-server environment, server processes are not required to maintain security-related state information for each client. This *stateless server* approach is generally accepted to be desirable for improving system reliability and scalability [16].

The dependence on timestamps for correctness, however, has the drawback that it assumes the presence of a globally accessible clock. While a global clock can be approximated by a system of synchronized clocks, the reliability and security of the underlying clock synchronizing protocol will have severe impacts on the availability of the system and the correctness of the authentication protocol itself. This makes the security strength of timestamp-based authentication doubtful, and is usually the main reason that such protocols are rejected by some system designers [1] [12].

3 Primitives for Inter-Process Communications

Distributed systems rely heavily on computer networks for the communications of data and control information between the computers of which they are composed. In most distributed systems it is common to find two types of communication services, namely, connection-oriented and connectionless services. Both these communication services are desirable due to the requirements of the applications which they support.

A connection-oriented service provides a reliable end-to-end message delivery facility. Before any data is transmitted, a reliable connection is formed between the principals wishing to communicate. This process involves the principal wishing to establish a connection sending a request to the recipient, and the recipient replying with its acceptance or refusal of the connection. In essence a "handshake" is performed between the two parties. In the midst of the handshake process, both ends of the connection establish state information for the reliable transmission mechanism.

Once a connection is formed a reliable exchange of information can be initiated. An example of a reliable connection-oriented service is the *Transmission Control Protocol* (TCP) which is part of the Internet Protocol Suite. A typical situation in which a reliable connection-oriented service is appropriate is when a large file is to be transferred from a remote site, since the time required to establish a reliable connection is usually not significant compared to the period elapsed during the actual data transfer. Moreover, most users of file transfer programs need to ensure that the file the destination receives is identical to the original copy. This type of service is also essential for the support of applications where the source and destination are required to interact constantly over a relatively long period of time, such as remote login and distributed window applications.

The provision of a reliable connection-oriented communication service, however, may not be preferred for every situation. In fact, there are certain distributed applications which find such a communication primitive undesirable, because its high reliability guarantee is not necessary while the overhead and delays incurred through the handshake are too costly. Connectionless communication services are usually provided to meet the requirements of such applications.

A connectionless communication primitive provides an application with the mechanism to transmit units of data across the network. Each data unit, which carries the full destination address and is routed through the system independent of all the others, is usually referred to as a datagram and is treated as an individual entity by the underlying network. Connectionless services commonly do not guarantee delivery and may transmit datagrams out of sequence or even duplicate them. An example of a connectionless communication service is the *User Datagram Protocol* (UDP) of the Internet Protocol Suite.

Connectionless communication services are sometimes preferred because of

4 Timestamp-based Security

As we have discussed, timestamp-based authentication is very attractive in that it eliminates the necessity of multi-step handshake and requires no extra state information to be maintained by the pair of communicants. However, its assumption concerning the use of a globally accessible clock hampers its popularity, since this global clock must be highly available, reliable and secure.

A reliable global clock can be approximated by using the individual processor clocks and requiring each processor to bring their clock values close to each other by means of some fault-tolerant clock synchronization protocol [10] [15]. Each individual processor is said to implement a local time server. A distributed implementation of the clock service reduces the reliance on a single component of the system, and thus makes the clock service more available. The clock synchronization protocol itself must be fault-tolerant so that the clock values on each correct processor are reliable in the face of network and other processor failures.

Keeping clocks in a distributed system synchronized without appealing to a single, centralized, time service requires that *clock values be exchanged and clocks periodically adjusted.* The time servers must provide accurate and precise time, even with relatively large stochastic delays on the transmission paths. Therefore, reliable time synchronization requires carefully engineered selection algorithms and the use of redundant resources and diverse transmission paths.

While the fault-tolerance of clock synchronization algorithms has been studied extensively in the past, little progress has been made to ensure their security. This seemed to be a consequence of the assumption about the environment within which existing clock synchronization protocols were developed to operate. A system of synchronized clocks was traditionally developed for the implementation of time-based synchronization of concurrent programming. Another main application of clock synchronization was in real-time process control systems which require that accurate timestamps be assigned to sensor values so that these values can be correctly interpreted. The environment within which these applications operated was usually assumed to be secure, while the system was expected to experience random failures, be they results of hardware or software causes.

In order to be used by a security service, the clock synchronization protocol itself should be secure. Most of the controversy regarding the use of timestamp-based authentication arose from its reliance on a system of synchronized clocks. A clock synchronization protocol can be made secure if all interactions among the time servers are protected by cryptographic means. The Signature-based Byzantine Clock Synchronization Protocol discussed in [8] is one such example. One of the consequences of using cryptographic techniques for securing clock synchronization is that performance (which is usually described in terms of the accuracy and precision of the synchronized clocks) may be degraded because encryption and decryption operations introduce extra delay in every communication path. The design of a secure clock synchronization protocol is an important area for

their simplicity and low protocol overhead. They also leave more room for application programmers to make their own arrangements for passing messages. A prime example to illustrate this point is *Remote Procedure Call* (RPC) [2], a widely accepted paradigm for interprocess communication in distributed systems. Most RPC subsystems are based on a connectionless datagram network service. The reasons for preferring datagrams are:

1. RPC messages are generally short and the establishment of a connection is an undesirable overhead.
2. The *Request-Reply* style of interactions involved in RPC can simply be implemented by a pair of datagrams: the datagram containing a request travels from the caller to the callee and the datagram containing the reply travels in the opposite direction.
3. Server computers may need to serve a large number of clients and the storing of state information relating to connections is undesirable.

Therefore, distributed programs using the RPC mechanism communicate by means of datagrams, but rely on the RPC subsystem to implement the semantics of remote procedure calls.

With regard to the design of secure distributed systems, mechanisms that protect all sorts of interprocess communications are of utmost importance. In a distributed system, one cannot ignore the possibility that messages within a distributed computing system may not come from where they appear to come from, and may not arrive at the place as they are intended to. Moreover, every packet of data being transmitted over the network is vulnerable to eavesdropping and tampering.

Distributed authentication is usually used together with interprocess communication mechanisms to achieve data secrecy and authenticity. Hence the authentication services of a distributed system should be provided in such a way that applications making use of them achieve improved security, and yet do not necessarily experience a significant degradation in their other qualities e.g. reliability and performance. Applications with different communication requirements should find the overhead incurred by the authentication service acceptable.

Hence, it is necessary to make both challenge/response- and timestamp-based authentication mechanisms available in a system, rather than just one of them, since both connection-oriented and connectionless communication services are desirable to satisfy the requirements of different applications. As discussed, the exclusive use of challenge/response-based authentications hampers some applications that, for performance reasons, were designed to heavily rely on connectionless communications. However, the security of solely using timestamp-based authentications is questionable.

further research, because if the cryptographic technique is not used carefully, the performance of the clock synchronization protocol will not be acceptable to most applications because of the extra delays.

Another interesting problem is to distribute the corresponding keys for cryptographic operations. A clock synchronization protocol will not be more secure by simply using encryption and decryption; the ways that cryptographic keys are distributed have a paramount effect on the security achieved. Key distribution, however, is usually achieved as a result of distributed authentication. This seems to suggest that the philosophy of building an authentication mechanism whose correctness relies heavily on a "to be secured" clock synchronization protocol is questionable, because the reliance relationship is a mutual one. Fortunately, as we shall see, this apparent paradox can be resolved.

The use of timestamp-based authentication is not as vulnerable as most people first thought. With the provision of both challenge/response-based and timestamp-based authentication mechanisms in a system, it is possible to retain the advantages of both schemes without keeping the disadvantages. The key point is to note that interactions among the set of time servers are constant and over a very long period of time. This suggests that a challenge/response-based authentication mechanism, which does not assume the existence of a globally accessible clock, is appropriate for clock synchronization protocols.

Our prototype adopted the approach of using a primary time server which periodically broadcasts its clock values to other machines. This approach is attractive because of its simplicity. Further, broadcasting is acceptable to environments within which timestamp-based security is needed. The reason being that timestamp-based authentication is most suitable to applications that employ connectionless communications; and connectionless communications, because of the lack of reliability guarantee, are mainly used by LAN applications. Hence the main purpose of timestamp-based authentications is to support the security needs of LAN applications. In addition, system overhead results from the use of broadcast in a LAN environment is usually acceptable.

In order to ensure that distributed clocks are closely synchronized, the primary clock needs to distribute its knowledge of current time frequent enough to avoid individual clocks drift too far apart. According to a preliminary measurement conducted in our system, a pair of clocks drift away from each other at a rate of one second per day. Therefore, clocks are resynchronized once every 60 seconds.

Such an approach to clock synchronization requires some kind of protection, however. Broadcast from the primary time source is exposed to a variety of security threats. In the context of secure clock synchronization, masquerading of time server, modification and replay of resynchronization messages are of our major concern. Fortunately, standard security services exist to guard against these kinds of attack. Indeed, a key distribution protocol which makes use of challenge-response is used to make the clock synchronization secure.

This idea is illustrated in Figure 1. The approach is to, first of all, provide

a challenge/response-based authentication mechanism as a fundamental component of the security service. A secure clock synchronization protocol, which relies on the provision of the multi-step handshake authentication, is then built to support a timestamp-based authentication mechanism. Each processor ensures the availability of an approximated globally accessible clock by executing a copy of the secure clock synchronization protocol. This protocol runs independently of other applications and works as an ordinary operating system service to supply timestamps to other system components. If a timestamp-based authentication service is invoked before the secure clock synchronization protocol starts, the system simply returns an error message notifying the event of "Security Service Not Initialized". A more rigorous design of the secure clock synchronization protocol is being developed [14].

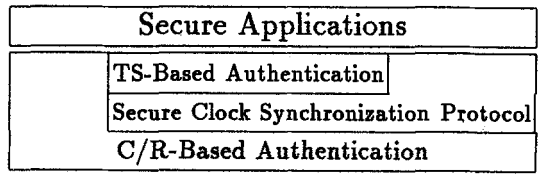

Fig. 1. Architecture for a Security Platform

Nevertheless, the foundation of our system security is still multi-step handshake authentication. In principle, at least, building a secure clock synchronization protocol is *not* impossible if a challenge/response-based mechanism is used; and once security of the system of synchronized clocks is ensured, our timestamp-based authentication can safely rely on it.

Therefore, the main purpose of providing a timestamp-based mechanism is not to enhance system security, but to complement the use of the challenge-response mechanism for those applications which find the multi-step handshake unacceptable. The system will be as secure as if timestamp-based authentication were not available, but its presence will encourage application designers to handle the security of connectionless communications more seriously.

Certainly, timeliness of authentication is only achieved at the cost of extra interactions among the legal participants of the authentication protocol. If a challenge/response-based approach is used, extra interactions (i.e. the multi-step handshake) are required before secure communications start. Whereas, if a timestamp-based approach is employed, although extra interactions are still needed, these extra interactions are transparent to the application programmer and, more importantly, do not necessarily introduce a significant communication overhead to the application before secure communication starts. This is because, with a timestamp-based authentication scheme, the extra interactions are accomplished by all participants of the secure clock synchronization protocol.

Before we close our discussion, it is worth noting that most of our current effort is on the design of a message logging mechanism for security purposes. As discussed in [7], timestamp-based authentication protocols, as they stand, do not provide a very secure way of detecting message replays. At the cost of extra storage, it is possible to eliminate the risk of accepting replay messages whose timestamps are still regarded as valid. In order to achieve this, the recipient may use a *logging mechanism* that records all messages received within the current acceptance window. Hence, freshness can be guaranteed by rejecting subsequent occurrences of identical messages within that window. The storage requirement of the logging mechanism is determined by the size of the acceptance window, which in turn is determined by the performance of the secure clock synchronization protocol and the expected message delivery delay. In principle, freshness assurance of timestamp-based authentication protocols should be provided by the logging mechanism [6]. However, a secure clock synchronization protocol is needed to reduce the size of the acceptance window so as to make the logging approach practical.

5 Conclusion

We have discussed the two classes of authentication protocols with respect to freshness guarantees. We have also looked into the features of two major mechanisms for interprocess communications. We suggested that a secure distributed system should make both kinds of authentication protocols explicitly available. An important advantage of this proposal is that both connection-oriented and connectionless communication services can be made secure without experiencing a significant protocol overhead.

This paper also argued that the use of timestamp-based authentication is not necessarily a security compromise as most people thought. We have discussed the security strength of timestamp-based authentication protocols, and presented one possible approach to achieve this.

Our proposal differs from the others in that we suggested the provision of both authentication mechanisms in a system and, more importantly, we believe that it is possible to build a timestamp-based protocol that is as secure as a challenge/response-based one. The use of nonce identifier and handshake has been suggested by Needham and Schroeder [11] [12] to ensure timeliness. This approach was criticised by Denning and Sacco [3] based on the ground that even a system of manually synchronized clocks could be sufficient to generate timestamps of the required quality. In fact, Denning and Sacco recommended the inclusion of timestamps in all authentication protocols. The timestamp-based approach was employed by project Athena of MIT, and the design of the Kerberos authentication system was based on it [9]. Regarding the timely property of Kerberos, various security weaknesses of its timestamp-based authentications have been discussed by Bellovin and Merritt [1]. In addition,the authors of [1], while having accepted that Kerberos's timestamp-based protocol be secure enough in

the Athena environment, suggested the provision of a challenge/response option to enhance security for a more general computing environment.

References

1. Bellovin, S.M., Merritt, M.: Limitations of the Kerberos Authentication System. ACM Computer Communications Review **20(5)** (1990) 119–132
2. Birrell, A.D., Nelson, B.J.: Implementing Remote Procedure Calls. ACM Trans. on Computer Systems **2(1)** (1984)
3. Denning, D.E., Sacco, G.M.: Timestamps in Key Distributed Protocols. CACM **24(8)** (1981) 533–536
4. Horster, P., Knobloch, H.-J.: Protocols for Secure Networks. Proceedings Eurocrypt'91, Springer LNCS **547** (1991) 399–408
5. ISO/IEC: Information Technology - Security Techniques - Entity Authentication Mechanisms - Part 1: General Model. ISO/IEC 9798-1
6. Lam, K.-Y.: Replay-Tolerance of Distributed Authentication. E.I.S.S. Technical Report (in preparation)
7. Lam, K.-Y., Gollmann, D.: Freshness Assurance of Authentication Protocols. Proceedings ESORICS'92, Toulouse, (1992)
8. Lamport, L., Melliar-Smith, P.M.: Byzantine Clock Synchronization. ACM Operating Systems Review **20(3)** (1986) 10–16
9. Miller, S.P., Neuman, C., Schiller, J.I., Saltzer, J.H.: Kerberos Authentication and Authorization System. Project Athena Technical Plan Section E.2.1, MIT (July 1987)
10. Mill, D.: Internet Time Synchronization: the Network Time Protocol. RFC 1129 (October 1989)
11. Needham, R.M., Schroeder, M.: Using Encryption for Authentication in Large Networks of Computers. CACM **21(12)** (1978) 993–999
12. Needham, R.M., Schroeder, M.: Authentication Revisited. ACM Operating Systems Review **21(1)** (1987) 7
13. Otway, D., O. Rees, O.: Efficient and Timely Mutual Authentication. ACM Operating Systems Review **21(1)** (1987) 8–10
14. Salkield, T.J.: Secure Network Time Synchronization. Ph.D. Thesis Proposal, Computer Science Dept, Royal Holloway, University of London (1992)
15. Schneider, F.B.: A Paradigm for Reliable Clock Synchronization. Proceedings of the Advanced Seminar on Real-Time Local Area Networks (1986)
16. SUN MicroSystems, Inc.: Networking Programming. (May 1988)
17. Tardo, J.J. , Alagappan, K.: SPX - Global Authentication Using Public-Key Certificates. Proceedings of the IEEE Symposium on Security and Privacy (1991) 232–244
18. C.C.I.T.T.: The Directory - Authentication Framework. C.C.I.T.T. (December 1988)