A Chat Room Assignment for Teaching Network Security

W. Garrett Mitchener
Applied & Computational Math
Princeton University
Princeton, NJ 08544
wmitchen@princeton.edu

Amin Vahdat
Department of Computer Science
Duke University
Durham, NC 27708
vahdat@cs.duke.edu

Abstract

This paper describes a chat room application suitable for teaching basic network programming and security protocols. A client/server design illustrates the structure of current scalable network services while a multicast version demonstrates the need for efficient simultaneous distribution of network content to multiple receivers (e.g., as required by video broadcasts). The system also includes implementations of two security protocols, one similar to Kerberos and another based on public key encryption.

1 Introduction

As the Internet continues to grow in size and scope, security will be an increasingly important aspect of overall system design. Unfortunately, the principles behind secure system design are typically not part of mainstream undergraduate computer science curricula. Reasons for such omission include the complexity of security protocols, the mathematical rigor required to understand encryption algorithms, and an overall lack of understanding of the principals behind secure system design.

However, a number of accelerating trends point to a need for better understanding of security concepts. First, an increasing amount of personal information is stored in servers connected to the public network. Examples include bank account information and medical history. Next, businesses, consumers, and the military are relying upon the Internet to carry out important day-to-day activities, often involving sensitive information. Finally, an increasing number of malicious users are employing widely available software to intercept personal information such as passwords and credit card information as well as to mount denial of service attacks on popular network services.

In this context, it is important to train computer science undergraduates in the vast array of issues and technologies associated with building secure systems. In this paper, we describe a first step in this direction, a chat room application that performs authentication and authorization of participants and ensures the secrecy and integrity of transmitted messages. Thus, participants must be able to prove their identity before transmitting messages preventing anonymous transmission and spoofing of unsuspecting users. Further, a transmitted message is not subject to eavesdropping by third parties; it is only readable by the intended recipients. Finally, receiver is able to reliably detect messages that have been altered in flight.

Two different versions of the assignment demonstrate the principals behind public key and secret key encryption techniques. Based upon student experiences with these applications, other popular security technologies such as Pretty Good Privacy, Secure Socket Layer, and Kerberos can be discussed in class with analogy to the developed chat room applications. The chat room will eventually be available at www.cs.duke.edu/~vahdat.

2 Chat Room & Security Specification

In this section, we describe some of the general requirements of our distributed chat room application. The user interface should have a text field in which incoming messages are displayed as they arrive, and a place for the user to type outgoing messages. Each message should have the name of the sender and a small amount of text. Any number of people should be able to join in.

The basic chat room is implemented with two underlying models to show different networking techniques. The first is the client-server model, in which a central server is responsible for distributing messages. Each client opens a TCP/IP connection to the server for sending and receiving messages. The server listens to each client with a separate thread of execution to avoid the case where a single poorly-behaved client disrupts all the others. This model illustrates the use of reliable, two-way data streams and the client/server architecture which underlies the majority of distributed services.

The second model uses multicast [2] to, in effect, eliminate the need for a central server in the architecture. Here, clients efficiently transmit messages directly to each other through a multicast IP address with the routers essentially replacing the server. As shown in Figure 1, a server can become a bottleneck if there are many users chatting at once; a multicast architecture avoids this problem and scales more easily. This model illustrates the use of multicast both to increase network efficiency (in terms of latency and consumed bandwidth) and to simplify program structure.

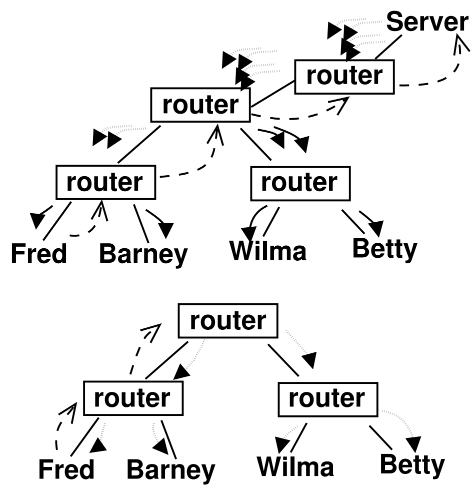

Figure 1: The top shows a server-based chat room. When Fred sends a message, it must be sent to each user individually, so it crosses some network connections multiple times. The bottom shows a multicast chat room which avoids this problem.

Although an insecure chat room may be suitable for casual conversation, many users desire some assurance that their conversations are private and cannot be intercepted by untrusted third parties. There are four specific requirements for a security system. First, it must guarantee the identity of each user, a process called *authentication,* which ensures that intruders cannot pose as other users and forge messages in their names. Second, the system must grant privileges to each user and enforce these privileges, which is called *authorization.* Third, the system must ensure the *integrity* of information, meaning any alterations made to a message must be detectable, thereby preventing an intruder from secretly corrupting a message. Finally, the system must enforce the *secrecy* of data to ensure that no one can eavesdrop on private conversations.

In designing a security protocol, the network is assumed to be insecure, meaning anyone can break in, read and alter data packets as they are transmitted. Privacy must therefore be based on encryption. Most terminals have only a keyboard and mouse for input devices, so retina scans and other means of user authentication are not considered; only passwords and confidential files are feasible means of authentication. The security protocol should also be easy to use and provide interactive performance comparable to insecure alternatives, otherwise the rate of adoption of the secure system will be hampered. Finally, the security protocol should be provably secure. In [1], the authors describe a formal notation and set of assumptions which describe a protocol's security. We leverage this logic in our protocols.

3 Security Protocols and Encryption

Encryption can be used to implement a security system. Two kinds of ciphers are commonly used: secret key and public key. Both use a block of data called a *key* to transform text in such a way that it is unintelligible to anyone who does not have the means to reverse the transformation. A *secret key* cipher uses one key for both encryption and decryption. A *public key* cipher uses two keys; any data encrypted with one key can only be decrypted by the other key. These two types of ciphers provide privacy and authentication in different ways. Entities using encryption for communication are called *agents.* We use the following notation: $\{\text{text} : \text{K}\}$ means text encrypted with key K. K* denotes a public key and K# the corresponding private key.

If a secret key is known only to two agents, they may use it to send encrypted messages to each other such that third parties will be unable to read them. If an agent sees a message encrypted with such a key, she may conclude that either she or her partner wrote it. Based on its contents, she should be able to tell which one of them wrote it, and when. She can therefore detect whether the message is a copy of one sent long ago, perhaps recorded and replayed by a malicious third-party agent. Thus, secret key ciphers can both keep a message private and guarantee its origin. The secret key must never be revealed and should be impossible to guess.

Public key ciphers work differently. Each agent has a pair of keys, one of which is publicly available, the other of which is known only to him. If one agent, Bart, encrypts a message with the public key of another agent, say Lisa, then only Lisa will be able to read it. If Bart encrypts a message with his private key, anyone can get his public key and decrypt it, but they will know Bart wrote it because only his private key could construct such a message. Thus public key ciphers can also be used for privacy and authentication. Private keys are typically very hard to guess, but must still be kept secret. Mappings between public keys and individual identities must be available from a trusted third party whose identity must also be authenticated (e.g., an identity server with a well-known public key).

An alternative means of authentication is a piece of data called a *nonce*. If only two agents know a nonce, then any message they see which includes the nonce must have been written by one of them. Another property of the message must be used to determine which agent sent it and when. If Bart and Lisa share a nonce, then Bart can send a message containing it to Lisa encrypted in her public key, and she may conclude that Bart sent it and no one else could read it. Without the nonce, Bart would have to encrypt the message with his private key and again with Lisa's public key to make it both authentic and private. The nonce avoids this slow double encryption.

In any protocol, agents must agree to trust certain servers, which are responsible for generating new secret keys and nonces. Messages from these servers must also be authenticated. Both types of ciphers also provide integrity. If an encrypted message is altered in transit, it will decrypt to nonsense, thus alerting the receiver that something has gone wrong.

4 Design and Implementation

The chat room demonstrates two security protocols, one based on secret key ciphers, and another based on public key ciphers and nonces. These two implementations illustrate alternative ways of constructing secure distributed applications and the use of the logic of authentication [1] to precisely describe the level of security they provide.

4.1 Secret Key Security Protocol

The chat room includes an implementation of a security protocol based on secret key encryption which can be used to give a hands-on illustration of how the popular Kerberos [3] protocol works, and the amount of planning that must go into designing such a protocol. When designing a security protocol based on secret key ciphers, there are several things to keep in mind. It is convenient to authenticate agents based on a password. The password is hashed to produce a secret key. If this key is known only to the agent and one server, they can send secret authentication messages to each other. For maximum security, the key must not be given to any other agent (including another server), nor can the password ever be sent in plain text over a network. Further, the number of messages encrypted using this key should be minimized. These precautions reduce the chance that a malicious agent will be able to read the password or collect enough data to guess the secret key.

If agents are to communicate securely but with minimal use of their password keys, then some other reliable source of secret keys must be provided. Once two agents are assured of each others' identities, they can encrypt their messages with a *session key*, which must be known only to them, come from a trusted server, and be used only for a short time and then replaced. They agree on the key by exchanging a piece of data called a *ticket* as follows. Consider the example where Bart and Lisa wish to employ a session key for private communication. They both trust Marge and have secret keys, KB and KL, known only to themselves and her. Lisa asks Marge to generate a session key. Marge constructs a ticket {Lisa Bart KS time} containing a new secret key KS and the current time. She then makes a two-part message {ticket {ticket : KB} : KL} and gives it to Lisa, who can decipher the message and extract the session key and time stamp from the ticket. Lisa believes the message is indeed from Marge because it was encrypted with KL, which is known only to herself and Marge. The time stamp proves it is fresh and not a replay of an old message. She sends the second part of the message {ticket : KB} to Bart. He reads it and extracts the session key, and believes it was from Marge because it was encrypted with KB. Both Bart and Lisa now know the session key KS and can use it to communicate securely.

A typical chat session using the secret key protocol, which requires two ticket exchanges, is illustrated in Figure 2. The agents are Bart, Lisa, the authentication server, the ticket server, and the chat server. Their password keys are KB, KL, KA, KT, and KC, respectively. Marge the administrator starts the authentication server, which has access to all the users' password keys and KT, and distributes session keys for the ticket server. She then starts the ticket server, which has access to the secret keys of the authentication server and all the other servers such as the chat room, and constructs session keys for them. These are the two trusted servers used by the protocol. Finally, she starts the chat server.

Bart starts a chat client, which contacts the authentication server and a ticket exchange takes place. The authentication server sends Bart a ticket containing a session key, K1. Bart sends the second copy of the ticket to the ticket server. Bart and the ticket server can now communicate securely using K1. Next, Bart asks the ticket server for a ticket to the chat server and another ticket exchange takes place. The ticket server sends Bart a ticket with a second session key, K2. He sends the new encrypted ticket to the chat server, which can decipher it to obtain K2. Bart and the chat server can now communicate securely using K2. Lisa goes through a similar procedure and can communicate securely with the chat server. Lisa and Bart can now chat securely, and only users with passwords to the authenti-

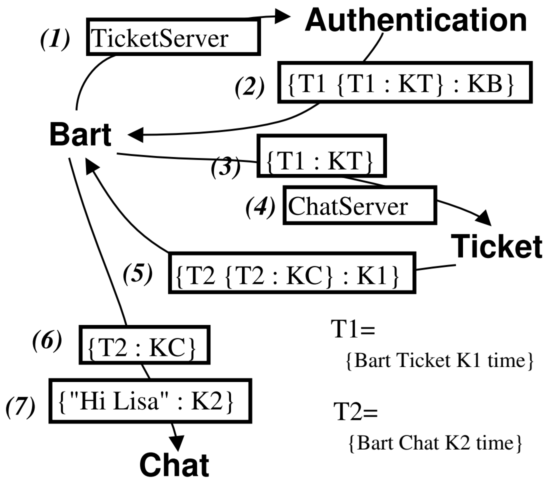

Figure 2: Example session of the secret key protocol.

cation server can join in. Since all messages to the chat server are encrypted with session keys, it can authenticate them, and relay that information to all clients.

The ticket server merits further explanation. The protocol could be designed so that each client receives its session key for the chat server directly from the authentication server. A more complicated client such as a network file system may need a session key for each individual file server, which would require the exchange of multiple messages with the authentication server encrypted with the user's password. With a ticket server, the protocol uses only a single such message, making it less likely that someone can gather enough data to guess the password key. In Kerberos, the ticket for the ticket server is stored in a temporary file when the user logs in and all of his client programs use it. The session key used to communicate with the ticket server has a short timeout (e.g., eight hours), requiring the user to periodically refresh this session key. The presence of a ticket server thus limits the exposure of the user's password.

4.2 Public Key Security Protocol

The chat room also includes an implementation of a second protocol using public key encryption. It illustrates the flavor of authentication techniques employed by existing Internet services such as e-commerce firms and online brokerages. This protocol requires fewer messages than the secret key protocol because it does not use session keys. Private keys are generally much harder to guess than secret keys, making it less important to prevent private key exposure. On the other hand, public key cryptography is on the order of ten times slower than corresponding secret-key protocols. Thus, in practice, many Internet services employ a public key exchange to establish a shared (session) key.

A sample session of the public key protocol is depicted in Figure 3. The agents are Bart, Lisa, the authentication server, and the chat server. Their key pairs are KB, KL, KA, and KC, respectively; * denotes the public key and # denotes the private key. The administrator, Marge, distributes a copy of the authentication server's public key, KA*, to all agents. She then starts the authentication server which has access to all agents' public keys, constructs nonces, and is trusted by all agents. Finally, Marge starts the chat server.

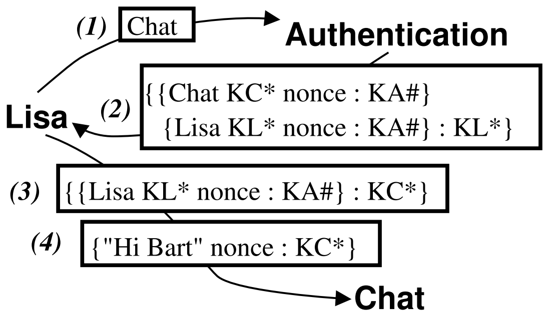

Figure 3: Example of the public key protocol. Time stamps are omitted.

Lisa starts a chat client, which contacts the authentication server. It constructs a nonce and sends Lisa a reply, both parts of which are individually encrypted with KA# (guaranteeing the origin of the message). The entire message is encrypted with KL* so only Lisa can read it, keeping the nonce secret. Lisa deciphers the message, reads the nonce and KC* from the first part, encrypts the second part {Lisa KL* nonce : KA#} with KC* and sends it to the chat server. The chat server can reverse both layers of encryption to obtain the nonce and KL*. Lisa and the chat server now share a secret nonce and know each other's public key. They communicate privately by encrypting messages to each other in the recipient's public key; the sender also includes a copy of the nonce to authenticate each message's source. Bart goes through a similar procedure, allowing him to chat securely with Lisa through the trusted chat server. Only users with public keys known to the authentication server can join in.

As an additional refinement to this project, the nonce may be used as a secret key for secure communication, which makes chatting faster as secret key ciphers require less computation than public key ciphers.

4.3 Java Implementation

The chat room and security systems are written in Java. Their design is based around several interfaces (see Figure 4). The whole package, including security servers, tools, and client/server and multicast implementations, all with debugging output, is about 6000 lines of commented source code spanning 75 files from 9 to 258 lines in length. The GUI is confined to two files totalling 112 lines. The Client and SimpleGUI classes are used in each client implementation and handle exchange and display of messages. The Reader and Writer interfaces encapsulate the input and output of messages and are implemented by several classes which handle encryption and communication with the network.

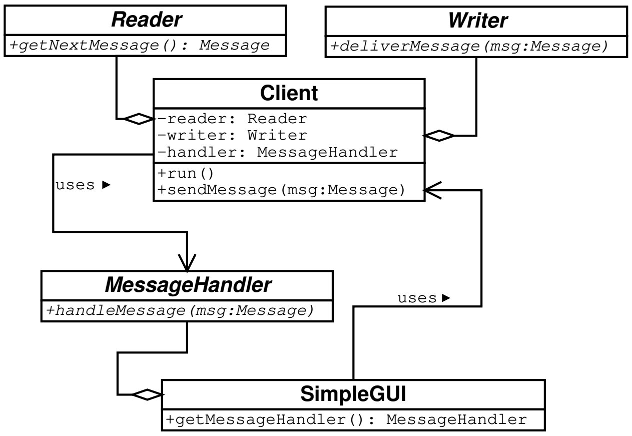

Figure 4: UML class diagram of the chat client.

The authentication and ticket servers handle clients sequentially, although they could be re-written to be multi-threaded. Each kind of message in each security protocol is represented by a class, and the Java serialization framework is used to transmit them. There are also utility programs for managing passwords and key files. The chat servers are multi-threaded with a separate thread for each client. The authentication procedure and encryption standard are encapsulated so that the code is flexible and re-usable.

5 Educational Use

This project attempts to fulfill a number of pedagogical goals at the advanced undergraduate level. First, the architecture of the application allows students to learn basic network programming, including the multicast service model and the structure of multi-threaded servers. Students also learn about the detailed architecture and implementation of the Kerberos security infrastructure. The public key version of the chat room illustrates the techniques employed by Internet services to carry out secure transactions over an insecure network. Finally, the performance characteristics of the multiple secure and insecure incarnations of the chat room illustrate the performance challenges faced by the designers of scalable network services. The package includes descriptions of several assignments for basic and secure chat rooms, the source code for sample implementations, and documentation for that code.

Possible programming assignments include asking students to re-implement the chat room and security systems individually or in groups. Since the security systems require several servers, tools which must work together, and well-defined interfaces, their implementation is well-suited to group work. If all the projects are coded to the protocols defined in the given implementation, they will be compatible. The compatibility and performance characteristics of each incarnation of the chat room can be demonstrated and measured by organizing a large chat session, spanning many computer labs, operating systems, implementations, and locations. To facilitate compatibility, the teacher could distribute some existing source code, e.g., the message classes, and make the existing implementation available in compiled form for use in testing.

Written assignments could be based on the logic of authentication. Students can be asked to use it to determine how the two security protocols work, exactly what level of security they provide, and to design and implement their own protocols. Such assignments could supplement lectures on Kerberos, PGP, and SSL.

6 Conclusions

This paper describes a chat room assignment designed to teach the basics of distributed programming and security protocols. A client/server implementation demonstrates the construction of many current Internet services while a multicast version illustrates the benefits of multicast to simplify system design and improve performance. Two different security implementations demonstrate the architecture of popular modern security protocols, including Kerberos and the public key architecture underlying the secure transactions provided by modern Internet services.

We are grateful to the NSF for supporting the development of this coursework (grant CISE 9634475).

References

[1] Michael Burrows, Martín Abadi, and Roger Needham. A Logic of Authentication. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pages 1–13, December 1989.

[2] Stephen E. Deering and David R. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. In *Transactions on Computer Systems*, May 1990.

[3] Jennifer G. Steiner, B. Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Proceedings of the 1988 USENIX Conference*, March 1988.