CS255: Cryptography and Computer Security **Winter 2004**

Programming Project #2

Due: Wednesday, March 10th, 2004, 11:59 pm

1 Overview

1.1 Introduction

For programming project 2 you will implement a chat room system with authentication using certificates and message transfer using SSL (Secure Socket Layer). The project 2 code is completely independent from that of project 1 (except for the GUI part). This project is larger in scope as compared to project 1, so please start early.

You will be learning

- **keytool (command line utility)** to generate and manage keys and certificates.
- **IAIK-JCE APIs** to create and sign certificates programmatically
- **JSSE (Java Secure Socket Extension)** to do secure networking.

1.2 Requirements

For this project, you will need to do the following :

- Secure all traffic using SSL.
- Build and use a public key infrastructure using X509 certificates.
- Use password authentication initially to procure the client certificates.
- Use SSL client certificates to successfully authenticate the client to the server.
- Use X509 Certificate extensions to provide access control for the clients, along with the authentication.
- Implement a secure and efficient online certificate revocation (user-banning) system. (extra-credit)

We will examine each of these features in detail below. Since we have not yet covered in the lectures all of the topics explored by this project, you may wish to start first on those aspects of the project that you can do immediately and save the other parts for later.

2 Description

2.1 Secure communication

Unlike project 1, you will be working will network sockets this time around. The JCE provides an abstraction for secure sockets in the java.net.ssl package and this relieves us from explicitly performing the key exchange, encryption and integrity of the messages transferred over these sockets.

2.2 Access control for clients

Every client which joins the Chat Server can be in one of the two chatrooms that the server supports. On the server side, these rooms are just logically separate lists of clients in each room, so that it does not interchange the messages from clients in different rooms. A client can be only in one of the two room. The access privileges for a client are encoded in the certificate it presents to the server during the SSL connection setup.

2.3 Setup of the system

The chat system now consists of three types of entities : chat clients, the chat server and the certificate authority (CertificateAuthority class). The Certificate Authority is an online entity which has an encrypted file containing the *usernames* and *passwords* of the expected chat clients. This is similar to what the server maintained in the project 1.*Passwords* are now stored after salting and hashing them, and verified in a similar manner. Along with the *username* and *password* for each client, the Certificate Authority also stores the access permissions for that client. To generate this encrypted file of *usernames*, *passwords* and *permissions* you can modify and re-use the file encrypter code you wrote for project 1.

When a client starts up it first connects (through SSL) to the Certificate Authority. Note that at this point the client utilizes no certificate for making the SSL connection. This means that the SSL connection provides only one way authentication of the Certificate Authority (CA) to the client. Now the client transmits its username, password and public key to the CA. The CA verifies the username and password (after salting and hashing appropriately). If this verification is successful, it generates a certificate, by signing the public key of the client with the CA's private key. This certificate contains the access permissions for the client as well.

The client uses the newly issued certificate from the CA, to connect to the ChatServer using SSL. Note that this time around the SSL connection will provide authentication in both directions and hence, you will not need any password based authentication between the client and the server. Based on the access permissions in the certificate presented by the client, the server will add it to the appropriate Chat Room.

2.4 Public Key Infrastructure

2.4.1 Offline Key Generation

The certificate authority has a public/private key pair which is generated offline using **keytool**. The **keytool** is used to generate a *keystore* for each entity in the system. Here is the sequence of actions which need to be performed before the chat system is bootstrapped.

1. Generate a public/private key pair for the certificate authority. The public key of the CA is self-signed.

2. Generate public/private key pairs for the ChatServer and each of the separate clients which will be joining the system.

3. Export the CA self signed certificate to a file and import it in all the other keystores.

4. Write a separate program which takes in ChatServer keystore and its associated password and signs the public key associated with the keystore with the CA private key.

At the end of the above steps you will have the ChatServer with a public key signed by the CA. All keystores will have the CA self-signed certificate, which is to indicate that everybody trusts the CA. Note that the ChatClient public keys are not yet signed.

2.4.2 Obtaining client certificates

On startup, a Chat Client connects to the CA and verifies its username and password with the CA. As indicated above, this SSL connection does not verify any certificates of the client and hence the password based authentication for the client is required. Now the CA creates and signs a certificate for the client (the X509CertificateGenerator class) and sends it over the SSL connection to the client. The CA uses certificate extensions to encode the access privileges of the client.

2.5 Certificate Revocation

For *extra credit* you can implement certificate revocation along which should prevent the clients whose certificates have been revoked, from joining the ChatServer again. Enforce a policy in the system, so that all clients using the words like "bomb" will be *kicked out* immediately from the room. You need to provide the following functionality to make this *kick out* foolproof :

- The client should not be able to connect to the server again using that certificate. The certificate issued to the the client would expire in due time however, until it expires, the ChatServer should reject that client.

- The client should not be able to procure a new certificate from the CA once it has violated the rules in the chat room.

We will be looking for a solution which is space-efficient and obviously foolproof.

3 Implementation

As with the first programming project, we have provided you with starter code. The starter code illustrates the basic socket and thread programming. See the following section for links of tutorials on socket and thread programming. In addition to Sun Java JCE library, you need IAIK JCE extension library to create and sign X509 certificates. The library is in the directory /usr/class/cs255/lib and it is also included in the starter code.

3.1 Description of the code

Here is a brief description of some of the starter code. The files you need to modify are in **bold** :

Makefile	Makefile for the project; modify this file to compile new classes that you add.
Chat/ChatClient.java	Request chat certificates from the CA use it connect to the Chat Server.
Chat/ChatServer.java	Accept secure connections from the clients.
Chat/ChatServerThread.java	Receive messages from the clients and post messages to the appropriate chat room.
Chat/ChatClientThread.java	Receive posted messages from the server.
Chat/ClientRecord.java	Stores client information.
Chat/ChatLoginPanel.java	GUI class for the login screen.
Chat/ChatRoomPanel.java	GUI class for the chat room screen.
Chat/CertificateAuthority.java	Starts up the Certificate Authority.
Chat/CertificateAuthorityThread.java	Accepts connections from the clients.
Chat/CertificateAuthorityActivity.java	Displays the activity of the client. Useful for printing debug output. Please make the output is relevant and succinct in the final submission.

Over and above modifying the above files, you will need to add a class which reads a file of client usernames, password and access privileges and generates an encrypted file using a key generated from the CA password. This class is run separately from the above Chat framework and is needed to

pre-compute the encrypted file which has a list of usernames and the corresponding authentication information.
You will also need to write separate class which signs the ChatServer's public key with the CA's private key and stores this signed certificate back in the ChatsServer keystore.

3.2 Running the code

You should spend some time getting familiar with the provided framework and reading the comments in the starter code. You will need to copy the /usr/class/cs255/project2/proj2.tar file to your account. As with project 1, you will also need to source /usr/class/cs255/setup.csh to set your path, classpath and java alias correctly. Building and running the Chat system is much the same as it was for project 1.

3.3 Crypto Libraries and Documentation

In addition to *java.security* and *javax.crypto*, some classes in *iaik.x509* and *iaik.asn1.structures* are also needed to do certificate management.
Important note: We require that your submission work with the Java API version on the Sweet Hall machines. Also, use the version of the IAIK library provided by us.
The following are some links to useful documentation :

- **Java API**
  http://java.sun.com/j2se/1.4.1/docs/api
- **IAIK-JCE API**
  http://jce.iaik.tugraz.at/products/01_jce/documentation/javadoc/index.html
- **Java Keytool Manual**
  http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html
- **JCE Reference Guide**
  http://java.sun.com/j2se/1.4/docs/guide/security/jce/JCERefGuide.html
- **JSSE Reference Guide**
  http://java.sun.com/j2se/1.4/docs/guide/security/jsse/JSSERefGuide.html
- **Sun Tutorial on Socket Programming**
  http://java.sun.com/docs/books/tutorial/networking/sockets/
- **Sun Tutorial on Thread Programming**
  http://java.sun.com/docs/books/tutorial/essential/threads/
- **IBM Tutorial on JSSE (Introductory)**
  http://www-106.ibm.com/developerworks/java/edu/j-dw-javajsse-i.html
- **IBM Tutorial on JSSE (Advanced)**
  http://www-106.ibm.com/developerworks/java/library/j-customssl/

Some classes/interfaces you may want to take a look at:

- java.security.SecureRandom

- java.security.KeyStore
- javax.net.ssl.KeyManagerFactory
- javax.net.ssl.KeyManager
- javax.net.ssl.TrustManagerFactory
- javax.net.ssl.TrustManager

- java.net.ServerSocket
- java.net.Socket
- javax.net.ssl.SSLSocket
- javax.net.ssl.SSLServerSocket
- javax.net.ssl.SSLSocketFactory
- javax.net.ssl.SSLContext
- javax.net.ssl.SSLSessionContext

- java.security.cert.Certificate
- java.security.cert.X509Certificate
- iaik.x509.X509Certificate
- iaik.x509.V3Extension
- iaik.asn1.ASN1Object

4 Miscellaneous

4.1 Questions

- We strongly encourage you to use the class newsgroup (su.class.cs255) as your first line of defense for the programming projects. TAs will be monitoring the newsgroup daily and, who knows, maybe someone else has already answered your question.
- As a last resort, you can email the staff at cs255ta@cs.stanford.edu

4.2 Deliverables

In addition to your well-decomposed, well-commented solution to the assignment, you should submit a README containing the names, leland usernames and SUIDs of the people in your group as well as a description of the design choices you made in implementing each of the required security features. For easier testing, please include a sequence of steps which will be required to run your system. Also provide all the keystores you have created and list their names and passwords in the README.
When you are ready to submit, make sure you are in your *project2* directory and type /usr/class/cs255/bin/submit.