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

Programming Project #1

Due: Monday, February 9th, 2004.

1 Overview

For the first programming assignment you are provided with a fully working Chat system in Java. The system as provided is totally insecure, all the messages are transmitted in the clear. You will be required to add security features on top of the provided framework, using the Java Crypto Extensions. You are allowed to work in groups of 2 members.

The required security features are :

- Secure storage(password protected) of passwords corresponding to each client on the server.
- Encryption of all chat messages with a block cipher in CBC mode.
- Integrity check for all chat messages using Message Authentication Codes (MACs).
- Resistance to replay attacks by eavesdroppers.

We will examine each of these features in detail in the following section

2 Security Features

2.1 Secure storage of client passwords

The server maintains a list of clients which can join the chat room. A client is identified by a *client handle* (a.k.a username). The server has a mapping from the *client handle* to the *client password*. This information is pre-generated (before running the Chat system) and can be read in by the server during initialization . The server stores the *client handle* to *client password* mapping in an encrypted file. The key to this file is generated using an admin password. In code, the server consists of two modules : a normal chat server program which accepts connections and serves clients, and an administrator program which pre-generates an encrypted file consisting of *client handles* and the *client passwords*.

When a client joins the chat server, the *client handle* is passed to the server. The server looks up *client handle* and gets the password for that client. Now the server and client can start sending encrypted messages to each other using a key derived from the password. Effectively, the *client password* becomes the shared secret between the client and the server. Note that the client's password is never sent on the network.

2.2 Message Encryption

Each message transmitted either by the client or the server must be encrypted using a block cipher. You may use any standard block cipher you like, but all the messages must be encrypted using CBC mode. The cipher key and should be generated from the *password* which the client and the server have. The CBC IV is generated at random for each message and sent along with the ciphertext.

2.3 Integrity Check using MACs

Every message going over the network should have a MAC, to enable detection of a malicious attacker tampering with the messages en route. Again the key for the MAC you decide to use, should be derived from the *password*.

2.4 Resistance to Replay Attacks

Even after you secure all the transmitted messages with encryption and MACs, there is still an obvious reply attack possible. An eavesdropper can capture a message en route to either the server or a particular client. He/she can then repeatedly send the message — which still a valid encrypted and MAC'd chat message — flooding the intended recipient and making the chat room unusable for other participants. Your solution should prevent the attacker from replaying a message on the server or the clients.

3 Implementation

You will be using the JCE (Java Cryptographic Extensions) while programming for this assignment. You should spend some time getting familiar with the provided framework.

3.1 Getting the code

Download the *proj1.tar* file linked on site to a directory in your account. Untar using the following command

tar -xvf proj1.tar

This should create the source tree for the project under the *proj1/* directory.

3.2 Description of the code

Here is a brief description of the files we provide. The files you need to change are in **bold**

Makefile	Makefile for the project; modify this file to compile new classes that you add.
java.policy	Policy file granting network/file permissions.
Chat/ChatClient.java	the Chat Client
Chat/ChatConsumer.java	Chat Consumer remote interface. Defines methods that can be called over the network by the server.
Chat/ChatConsumerImpl.java	Implementation of the ChatConsumer interface.
Chat/ChatLoginPanel.java	GUI class for the login screen.
Chat/ChatRoomPanel.java	GUI class for the chat room screen.
Chat/ChatServer.java	Chat Server remote interface. Defines server methods that can be called remotely by the client.
Chat/ChatServerImpl.java	Implementation of the ChatServer interface.

Over and above modifying the above files, you will need to add a class which reads a file of *client handles* and *client passwords* and generates an encrypted file using a key generated from the server admin 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 *client handles* and the corresponding *client passwords*.

3.3 Running the code

You will also need to *source /usr/class/cs255/setup.csh* to set your path, classpath and java alias correctly. If you are an advanced user, you may want to do this manually. To build the project, enter the *proj1* directory and type *make*. To run the Chat system, follow these four easy steps:

1. Pick a unique port number for your group to use. We recommend using the last four digits

of your phone number. Note this wont work if your last four digits are less than 1024 or your roommate is taking the class.

2. Start the RMI registry. This is part of the networking infrastructure we have set up for you. To start the registry, type

   elaine21:~/proj1> rmiregistry portNum &

   If you omit the port number, the registry will start on port 1099, so be careful.

3. Start the Chat Server. Note that the server does not necessarily need to be running on the same machine as the registry. You do, however, have to specify the host and port on which the registry is running. For example:

   elaine21:~/proj1> java Chat.ChatServerImpl elaine21 portNum &

4. Start one or more clients

   elaine21:~/proj1> java Chat.ChatClient &

Important note: In the past, the Sweet Hall people have been grumpy with CS255 students leaving the rmiregistry and ChatServer processes running after they logout. You need to explicitly kill these processes.

3.4 Crypto Libraries and Documentation

Javas security and cryptography classes are divided into two main packages: java.security.* and javax.crypto.*. They have been integrated into Java 2 Platform Standard Edition v 1.4. Classes for cryptographic hashing and digital signatures (not required for project 1) can be found in security, whereas ciphers and MACs are located in the JCE. The following are some links to useful documentation :

- Java API
  http://java.sun.com/j2se/1.4.1/docs/api
- JCE Reference Guide
  http://java.sun.com/j2se/1.4/docs/guide/security/jce/JCERefGuide.html
- Java Tutorial
  http://java.sun.com/docs/books/tutorial/
- Chapter 6 from Java Cryptography by Jonathon Knudsen
  http://www.oreilly.com/catalog/javacrypt/chapter/ch06.html

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

- javax.crypto.KeyGenerator
- javax.crypto.SecretKey
- javax.crypto.IvParameterSpec
- javax.crypto.Mac
- javax.crypto.Cipher
- javax.crypto.CipherInputStream
- javax.crypto.CipherOutputStream
- javax.crypto.SecretKeyFactory

- java.security.MessageDigest
- java.security.SecureRandom

- java.math.BigInteger

3.5 A Word About Networking

The Chat system would be rather boring if it didnt run over the network. The problem for us is that we cannot expect everyone interested in cryptography to have network programming experience. For that reason, we will be using Java RMI for all network communications in our programming projects.

RMI stands for Remote Method Invocation and it is a standard part of the Java library. RMI enables you to declare objects as remote, making the methods of that object accessible to Java programs running on different machines.

All of the RMI setup has already been done for you. For instance, in ChatClient.connect, you will find the following code:

```
_server = (ChatServer)Naming.lookup(// + host + : + port + /ChatServer);
```

This simply sets the variable `_server` to a reference to an object called ChatServer which has been registered with the rmiregistry running on the designated host and port. In ChatClient.sendMessage, you see the code:

```
_server.postMessage(msg, _clientID);
```

This has the effect of sending msg and `_clientID` over the wire to the machine where the ChatServer is running and calling the servers postMessage method with msg and `_clientID` as parameters.

If you want to add a new remote method, you need to do so in two places. You need to modify the ChatServers remote interface in ChatServer.java and the actual method definition in ChatServer-Impl.java. The same is true for the ChatConsumer.

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.

When you are ready to submit, make sure you are in your *proj1* directory and type /usr/class/cs255/bin/submit.