CS255: Cryptography and Computer Security **Winter 2003**

Programming Project #1

Due: Tuesday, February 25th 2003, 11:59pm

For the programming projects in this course, you will be working in groups of two. Your goal is to increase the security of the provided Chat system. Right now, the Chat system is totally insecure: all messages from the client to the server and from the server to the client are transmitted in the clear.

For project 1, the required security features are:

- A key exchange between the chat client and the chat server when the client connects.
- All chat messages need to be encrypted using a block cipher in CBC mode
- A MAC must be appended to all chat messages.
- You must devise and implement a measure to combat replay attacks.

We will examine each of these features in detail below.

Key Exchange

You may use any of the key exchanges discussed in class, including the Diffie-Hellman key exchange protocol or any scheme that uses public key encryption (we showed in class that a secure public key system implies a key exchange algorithm). Please no Merkle Puzzles. Note that we have not yet discussed a method to prevent the "person-in-the-middle" attack on key exchange protocols. Consequently, in this project you are only required to secure the key exchange protocol against a passive eavesdropper. We will deal with active attacks on the key exchange protocol in project 2.

You should use the secret obtained during key exchange to generate your block cipher key, your CBC IV and your MAC key. To do this, you should use a hash of the secret as input to a PRNG, and use the output of the generator as your keys and IV.

Message Encryption

Each message transmitted by either a client or the server must be encrypted using a block cipher. You may use any standard block cipher you like, but all messages must be encrypted using CBC mode. As mentioned above, you should generate both the cipher key and the CBC IV from the secret obtained during key exchange.

MACs

Every message that goes over the network should have a MAC to enable the detection of a malicious attacker tampering with messages en route. You should generate the key for this MAC from the secret obtained during key exchange as described above.

Preventing Replay Attacks

Even after you secure chat messages with encryption and MACs, there is still an obvious replay attack. Marvin captures a chat message en route to either the server or a particular client. He then repeatedly sends that message – which is still a valid chat message – flooding the intended recipient and making the chat room unusable for the participants. Your solution should prevent an attacker from replaying a user's message.

Security Holes

Even with these four security features, it should be clear that there are still holes in the Chat system. Aside from the "person-in-the-middle" attack on the key exchange protocol, the biggest remaining problem is authentication. The easiest way for an attacker to see the messages in the chat room is simply to join. Another problem (at least for the .com world) is that there is no way to charge people for using our chat system. We will address some or all of these problems in programming project 2.

Implementation

As mentioned in class, you will be programming this project in Java. We have provided a substantial amount of starter code. Here is a description of the files we provide for you (files you need to change are in **bold**):

File	Purpose
Makefile	Makefile for the project – modify this to build any classes you add to the project
java.policy	Policy file granting network/file permissions
Chat/ChatClient.java	Chat Client
Chat/ChatConsumer.java	Chat Consumer remote interface – defines methods that may 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 may be called over the network by clients
Chat/ChatServerImpl.java	Implementation of the ChatServer interface
Chat/SkipConstants.java	Contains constants for the 1024 bit prime p and generator g used for Skip. Use these if you don't have the patience to generate your own.

You should spend some time getting familiar with the provided framework. You will need to copy the /usr/class/cs255/proj1 directory to your account. 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 won't 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 Chat 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.

Crypto Libraries and Documentation

Java's 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 documentations.

Description	**URL**
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 you may want to take a look at:

javax.crypto.KeyAgreement
javax.crypto.KeyGenerator
javax.crypto.IvParameterSpec
javax.crypto.Mac
javax.crypto.Cipher
javax.crypto.SecretKeyFactory

java.security.MessageDigest
java.security.KeyPairGenerator
java.security.SecureRandom

java.math.BigInteger

A Word About Networking

The Chat system would be rather boring if it didn't 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.

What is Java RMI?

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 server's `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 ChatServer's remote interface in ChatServer.java and the actual method definition in ChatServerImpl.java. The same is true for the ChatConsumer.

Two cents: don't change RMI infrastructures unless you are expert. We don't want you to spend much time on network programming.

Help

- 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.

Submission

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.