CS 4235 / CS 8803IIS Homework 5

Assigned: 9 March 2011
Due: 28 March 2011, 5:00pm Atlanta time. Students submitting solutions after that time but by 5:00pm Atlanta time on 30 March will have their scores scaled by 0.8. No solutions will be accepted after 5:00pm on 30 March.
Teaming: This is a group project. Please work in a group of size two or three. You may choose your own group partners.

Your solution must be your own (and teammates) work. Although you can use outside sources for information and library code, you:

- **must** be legally allowed to use any code not written by you, and
- **must** cite all sources of information and code. A citation should provide sufficient information for myself or anyone else to find the source that you used.

We may employ the use of automated code analysis and comparison tools as part of our evaluation. Any suspected academic conduct violations will be referred to the Dean of Students Office. If you are ever unsure whether or not you are acting in violation of academic conduct policies, please ask us rather than guessing.

This homework has one programming part worth 200 points. Scores will be posted on T-Square.

This assignment is programming-intensive. If you do not get an early start, you may be unable to successfully complete the project. You may implement your solution in any of C, C++, or Java without prior approval from the TA. You may use a different language only if the TA gives you permission.

In this project, you will develop an instant messaging system called Chatter. Chatter has a client-server architecture that allows multiple clients to communicate through a single shared server. The human user of each client must successfully authenticate to the server, and all communication between every client and the server is encrypted to provide confidentiality from eavesdroppers.

The Chatter "system" is a single server and one or more clients connected to the server. You are responsible for implementing the complete system; this will require both network programming and cryptographic programming. For this project, all components of the system should execute on Red Hat Enterprise Linux as found on any of the killerbee systems (e.g. killerbee1.cc.gatech.edu).

Chatter server

The Chatter server is a command-line network server with two purposes:

1. It authenticates human users to the Chatter system.
2. It securely distributes messages typed by one user to all connected clients.

The server should have a database of known usernames and the cryptographic hash of each users password. You do not need to implement functionality to add or change existing users and can manually create the database. Note that passwords should not be stored in cleartext so that accidental disclosure of the database to an attacker does not allow the attacker to know users passwords. Choose a non-reversible cryptographic hash function, and store only the hash of each password. An attacker cannot reverse the hashes to determine passwords, but the Chatter server can still authenticate users by recomputing the hash of a typed password and verifying that it matches the stored hash in the user database.

When started, the server should begin listening for incoming TCP connections from Chatter clients. Upon receipt of an incoming connection, the server and client should use Diffie-Hellman key exchange to establish a shared secret key. Note that the D-H values chosen by the client and server (g, p, a, and b) should be selected using a random number generator so that each session will have a different key. As soon as the key is established, the client and server should begin sending all data encrypted with a symmetric key protocol using the D-H secret key.

After establishing the secure channel, the server should authenticate the human user of the Chatter client. The client should send the username and password of the user across the encrypted channel to the server. The server should recompute the password hash and then verify the username and hash against the database of known users. If the verification fails, then the server should disconnect the client. Otherwise, the server next provides instant messaging service to the client.

An authorized client sends messages to the server and receives messages back from the server (including messages sent by itself). The server should accept a message from any connected and authorized client. It should then distribute that message to all authorized clients, including the client that generated the message. Note that distribution will require the server to decrypt the message using the secret key shared with the sender and then to reencrypt the message using each clients unique shared secret key. When distributing messages, the server should also send the username of the client that generated the message.

A message is one line of printable characters ending in a newline. Messages may have an arbitrary number of characters before the newline, so be sure that your server and client allow arbitrary length strings.

The server should silently discard messages with unprintable characters.

Clients may come and go during while the server is providing instant messaging service, so it should be able to authenticate some clients while simultaneously receiving and distributing messages to other clients already authenticated.

For our testing, the server should support up to four simultaneously connected users. You may program this support into the server in whatever way is easiest for you. For example, the server can listen for all connections on one port and then pass off each incoming connection request to a child process. This approach scales nicely to a large number of simultaneous connections. Alternatively, the server could listen on four ports and expect each of the four clients

to connect on different ports. This will allow you to write a simpler single-threaded, single-process server, although it is less elegant. We suggest printing a message to the screen when the server starts up instructing us how to connect each of the clients to the server.

Once started, the server should be non-interactive. At your discretion, you can have the server print out debug messages if you think they will help us test your system, but we should not need to type commands into the server. We will expect to terminate the server's execution via Control-C.

The server should gracefully handle exiting clients or lost network connections. The server should simply remove that client from its list of currently connected clients, and new clients should still be able to connect to the server and join the system.

Chatter client

The Chatter client is a command-line interactive program that a human uses to chat with other participants of the Chatter system. The client should take command-line arguments that specify how to connect to the Chatter server. This should include the TCP port at which the server is listening, and may optionally include a computer name or IP address specifying where the server is executing. Note that our tests will execute the server and all clients on the same machine, so a machine name argument is not required.

When started, the client should attempt to connect to the server. If the attempt fails (perhaps the server is not executing), then the client should print an error message and exit. If the connection succeeds, then the client should run the Diffie-Hellman key exchange algorithm with the server to establish the shared session key. After establishing the key, the client should use symmetric key cryptography for all subsequent communication with the server.

The client should prompt the user to type a username and password at the keyboard. The client then sends this information to the server over the encrypted channel for user authentication. If authentication fails, then the client should print an error message and exit. Otherwise, the client should enter its instant messaging mode.

In the instant messaging mode, the client receives input from both the keyboard and the server. Input from the keyboard is made of messages typed by the human using the client. A message is a single line of printable text ending with a newline. When the user finishes typing a message (e.g. presses ⟨Return⟩), the client should send that message across the encrypted TCP channel to the server. Input from the server is comprised of messages and the username of each message's author. The client should print each message to the screen in the format:

⟨username⟩: ⟨message⟩

without the angle brackets. All messages from the server should be printed, including the messages that were sent by the same client. Make sure that you

can handle the case where the client receives a message from the server while the human user was in the middle of typing at the keyboard.

We will expect to terminate a client's execution by pressing Control-C, although you are free to create a different exit mechanism at your discretion.

The client should gracefully handle an unexpected termination of the server or a lost network connection. Printing an error message and exiting is a suitable behavior.

Cryptographic algorithms

You need to choose two cryptographic algorithms for use in the Chatter system:

1. The hash algorithm used on user passwords.
2. The symmetric-key cryptographic scheme used for data confidentiality.

Select algorithms that you believe appropriate based on our classroom discussions. You should not need to implement either of these algorithms. It is perfectly suitable to find and use libraries implementing cryptographic operations.

You may need to massage the data that you would like to encrypt. Algorithms often expect plaintext data of a specific length, so you may need to break long plaintext into smaller blocks and pad short messages out to a full block length. When decrypting a message, your code should remove padding and reassemble blocks as necessary.

Diffie-Hellman key agreement

You will need to use D-H key agreement to create the shared secret key used for subsequent symmetric-key cryptography. The algorithm was covered in class; recall that it uses the following mathematical operations:

- Selection of a large prime number.
- Exponentiation of a large number to a large exponent in modular arithmetic.

You are welcome to find and use a library providing this functionality. However, these operations are simple enough that you may choose to implement them yourself.

For this project, you should choose unsigned integer values less than 2^{32}. Beware of integer overflow: the square of a 32-bit number may require 64 bits to represent. If implementing your own algorithms, you should make use of the unsigned long long data type, or use a library for large numbers.

For efficient selection of a large prime, look up the "Miller-Rabin primality test" online.

For efficient large number exponentiation, look up "Exponentiation by squaring" online.

Network protocols

Your Chatter system should use TCP for all communication between clients and a server. The system requires several different types of data to be sent at different points of execution:

- D-H parameter exchange data (in cleartext).
- Data containing a humans username and password (encrypted).
- An authentication response from the server (encrypted).
- Data containing a username and message sent by a human to all participants in the system (encrypted).

Depending on how you implement connection establishment and connection teardown between a Chatter client and a server, you may have additional protocol messages not listed above.

You will need to develop the necessary protocols. We suggest creating a list of different message types, where each type uses a unique identifier in the first byte of the message. The recipient of data from the network can then read the first byte to know how to interpret the remaining data.

How to proceed

This project requires a fairly extensive implementation, so it is important to plan your solution. I suggest dividing the work among the team members. For example, one team member could work on the server and another could work on the client. They could initially ignore the need for encryption simply to get network communication operating properly. Meanwhile, the third team member could be developing the necessary cryptography which can then be easily inserted into the client and server code.

What to turn in

Submissions will be done electronically on the T-Square assignments page. Each group only needs to submit one solution, but it should list all group members.

Create a README file containing the following information:

- The names of all group members.
- A description of what you were able to complete. If you believe that you have a fully working solution, then this explanation will be brief. Otherwise, you should explain what we will find working and what we will find broken or missing when we begin testing.
- A statement indicating what cryptographic algorithm you used to compute password hashes.

- A statement indicating what cryptographic algorithm you used for symmetric encryption and decryption.
- A list of the usernames and passwords for the users in your user database.
- Exact, step-by-step instructions telling us how to build your Chatter server and client on a College of Computing Red Hat Linux server.
- Instructions telling us how to run your solution. Please give the exact command-lines that we need to type.
- A statement telling us which TCP port numbers are used by your system; or, instructions stating how we can figure this out at runtime (in case the ports are chosen differently for each execution). We need this information for part of our testing: we may use network monitoring tools to verify that the communication channels between clients and servers carry encrypted traffic.
- Citations to websites stating where you found any non-standard libraries used in your solution.

Create a zip file or tarball containing the following:

- All source code that you wrote.
- All library code that is non-standard (e.g. you downloaded it from the Internet). Without this code, we will be unable to build your utilities.
- A compiled version of your server and client all ready for us to run immediately.
- A user database file containing usernames and password hashes.
- The README file described above.

Post your zip file or tarball to T-Square. Only one group member needs to post the solution, but they should list their team members as part of the submission.