Project Description

CS 161 - Joseph/Tygar

September 27, 2006

Overview

The goal of this project is to design and implement a secure instant messaging program. Below are a set of functional requirements followed by security requirements of the program. All aspects are strict requirements of the program.

Deadlines

This project will be split into two parts. First, your group will write a design document describing how your system will work. **This design document is due on October 2nd at 11:59 pm via email to your TA.** Your group will meet with your TA to go over the design document, and suggest improvements if necessary. The second part will involve your group writing the code to implement your system, testing it, and writing the final document. **The final hand-in is due on October 16th at 11:59 pm.** The details of the final submission process will be provided on the course webpage at a later time.

Group Formation

You should choose your groups as soon as possible if you have not already done so. The design review deadline will come up quickly, so you need your group so that you can start working right away. All groups should be finalized no later than Monday, September 25th.

Your group should consist of four students in the same section (though there may have to be 1-2 groups of 3 or 5 students if the number is not a multiple of 4). Feel free to use the newsgroup to find partners, though be sure to say which section you're in. You can also let your TA find a group for you.

Each group or partial group should send an email to your TA with the following information for each person: name, email address, instructional login name, and ssh public key (see below). Please send this email by Monday, September 25th at the latest (earlier if possible, especially if you have need your TA to help you find partners).

SSH Public Key

We will be setting up a repository for version control for each group. As part of this process, you need to generate an ssh public key for your instructional account (this is different from generating a PGP public key with gpg). Please follow these steps on pulsar.cs.berkeley.edu (or another instructional machine, but pulsar is known to have the right software available).

1. Run '`ssh-keygen -t dsa`' from your named instructional account. The default filename `id_dsa` is fine.
2. Create a file called `identification` in your `~/.ssh` directory with the following line:

   ```
   IDKey    id_dsa
   ```

3. The file `~/.ssh/id_dsa.pub` is your public key, and this is what you should send to your TA. It should contain one line that starts with `ssh-dss` and then has a very long string of random-looking letters, numbers, and symbols, followed by a string like '`yourlogin@pulsar`'.

Instructions for setting up your repository will be provided on the course webpage in the near future.

System Functionality

The program must meet the following functionality requirements:

Account Each user has an account within the program. A user must use this account to perform messaging with other users. We will supply a list of accounts that should be in your system.

Messaging Any user can send a text message to any other user on his or her buddy list.

Buddy List The program must maintain a list of other users that are allowed to communicate with the user. These other users are referred to as buddies. The user can only communicate with buddies on his or her buddy list. Thus, a user must be added to the buddy list before a conversation takes place. A user may remove buddies from his or her buddy list. This will block all future messages from the removed buddy to the user. The exact definition of "block" depends on implementation, but at the very least blocked messages should not be displayed.

Conferencing The program must contain support for conferencing between buddies. The user can specify two or more buddies to participate within a conference, and only those in the conference can see the conversation taking place. Thus, the user who initiates the conference, known as the administrator, has the power to invite to the conference when the conference is created. Additionally, any user participating in a conference has the ability to remove himself or herself from the conference.

Implementation details

A skeleton IM client implementation based upon the Hamsam Java API will be provided. Each group will design and implement a secure network protocol to allow the clients to communicate. Specifically, each group will be implementing Hamsam's "Protocol" interface.

When you untar the code, the first thing you will want to do is run the GUI and take a look at what it offers. You can run it by typing

```
make launch
```

Type in anything for the username and password, and the skeleton implementation tells the GUI that you have logged on. You can add buddies, start conferences, and chat. Whatever message you type is relayed back to you. You will need to flesh out the underlying implementation such that it communicates with a server and with other chat clients, rather than merely looping back to itself. Now take a look at the code for this skeleton, which is found in

```
./hamsam/protocol/secure
```

Read the code and the comments. We do not provide any networking code whatsoever, you must write that yourself. Hamsam provides a GUI only, and the skeleton illustrates how to communicate with the GUI. It is up to each group to design their own architecture. Keeping in mind the functionality and security goals, as well as ease of implementation, you should consider the relative merits of a centralized server versus partially or fully decentralized peer-to-peer designs. We are not providing any code to get you started on the server side of things, this must be written from scratch, building on Java APIs.

Feel free to use any of the APIs provided with Java 1.5 to implement your project. If you come across any third party libraries that you think will be helpful, contact your T.A. Also, if you are uncomfortable using Java for this class, let us know.

You will need to write code to open network connections to remote hosts, and send and receive data over those connections. The standard way to do this is via the Sockets API. Java provides a simple interface to sockets, which you can learn about here:

```
http://java.sun.com/docs/books/tutorial/networking/sockets/index.html
```

Implementation Notes

When you click the Add Buddy button, hamsam provides a window where you can type in the buddy name. The Add Buddy window also asks you type in a group. You can ignore this. You can add any buddies you want, your buddy does not have to give approval before being added. However, if somebody is on your buddy list but you are not on theirs, then when you send a message to them, the message will be blocked.

The way to send a message to one person is to click on their name on your buddy list, then click 'Chat with'. To initiate a conference, shift-click or control-click to highlight multiple names, then click Chat with. While the user interface may appear similar when it comes to chatting with one person as compared to chatting with multiple, note that the underlying implementation is distinct. Chatting with multiple people is a conference, which involves different functions and objects.

The hamsam GUI runs in the same thread as the callback code that is activated when the user clicks on the GUI. This means that if an action is taken which will take a significant amount of time, such as waiting for the next message to come across the network, then the GUI will become unresponsive. A good way to fix this is to create a separate thread for receiving messages.

You are not expected to create a user interface for adding and removing accounts. It is fine to build a fixed set of accounts into your system. We will supply a list of accounts that must be in your system, and you are free to have more accounts as well.

Important Change In Security Assumptions

Assume that the machines on which your chat systems run are locally secure. This means you need *not consider* attacks where the attacker is logged on to or has physical access to the machines running the client or server. In particular, feel free to store passwords and secret keys in plaintext. Note that this is a relaxation on the security requirements compared to what was originally given. Assume that the attacker still has full control over the network, thus it is still not allowed to transmit secrets in plaintext across the network.

Security Requirements

The program must meet the following security requirements:

Authentication The owner of the account is the person who created it. Once an account is created, only the owner can access it unless the owner divulges secret information or otherwise disobeys the authentication protocol.

Confidentiality Sent messages can only be viewed by the buddy or buddies intended to receive them. Thus, no one other than the intended recipient(s) should be able to read messages from the sender.

Integrity Sent messages cannot be altered by a third party. All messages must be protected from modification between sender and receiver.

Cryptography APIs

As part of your project, you will need to use some cryptographic functions. While you are free to use any implementation you like, we recommend Java's built-in cryptography API, the JCE (Java Cryptography Extension). The JCE classes can be found in the package `javax.crypto`, and you might also explore `java.security` (both are included in J2SE 5.0).

The best place to start learning how to use the JCE is probably the "JCE Reference Guide":

```
http://java.sun.com/j2se/1.5.0/docs/guide/security/jce/JCERefGuide.html
```

The quickest way to get going will be to skip down to the Code Examples and refer to other parts, and the javadocs, as needed. Appendix A lists the names that you can choose from when selecting the algorithm, mode, etc.

If you want more information and specifications, look here:

```
http://java.sun.com/j2se/1.5.0/docs/guide/security/CryptoSpec.html
```

Or, for a lot more security-related information (most of which is completely irrelevant to this project):

`http://java.sun.com/j2se/1.5.0/docs/guide/security/`

Development and Testing

For your development and testing of this project, you will have access to the DETER network emulator and testbed (see `http://www.isi.deterlab.net`). Details on using this resource will be provided on the course webpage in the near future.

Design Review

Before implementing your design, your group will present its system architecture to your TA. First, each group will submit their design documentation to their TA. Please use PDF, Postscript, HTML, or plain text. Your TA will schedule a meeting with you shortly thereafter to review the design. The review will cover the Security Goals, Threat Model, and Attack Analysis that your group produced when designing the system, as well as the architecture that your group plans to implement. This will be an excellent opportunity for your TA to provide you with feedback on your design and answer any remaining questions your group may have.

Design Document Specification

Writing a design document will help you to thoroughly think your design through, before beginning coding. You should not write any code before the design document is complete and has been reviewed with your TA. However, you should look at the provided skeleton code first, as this will clarify what parts you need to write and what parts we provide.

Your design document should address the following topics.

Protocol Overview Describe your design at a very high level. How do systems in the IM network communicate with each other? What messages do they send and what is the purpose of each one?

Security Model Develop a security model for your system. Follow the outline provided during lecture 2. Consider the goals, motivations, and capabilities the attacker has. Discuss which attacks you are defending against, and which you are not, and why.

Module Descriptions The purpose of the design document is to force you to think analytically about your system. A standard technique for understanding systems is to break them down into modules. Note that a module is a conceptual tool to help you think in an organized way. It is not a Java construct such as a class or method. A module may correspond to a class, or several classes may make up one module. In a non-object-oriented system, a module might not be reflected directly in the language at all, the modules might just be groups of functions that work together in performing a task. A module is any part or subsystem of a larger system. A module may itself be decomposable into other modules that each provide part of the parent module's functionality and hide some of the parent module's secrets. A module decomposable into other modules is called an internal module; a module that is not decomposable into other modules is called a leaf module. Internal modules tend not to have internal computational structure, serving instead as grouping mechanisms for their component modules. Leaf modules, conversely, have a rich internal computational structure. Certain information is relevant for every module in a module description, but some information may not be relevant for all modules. Use your discretion about how much information is necessary to help us understand your design. Information that may be relevant for your modules includes the following:

Module Behavior Explanation of a module's behavior includes (1) a catalog of the operations it provides, with descriptions of their input and output and their externally visible effects, and (2) a catalog of the data types and objects that the module provides.

Assumptions A module's assumptions are the conditions that must be satisfied for an implementation of the module to work. For example, a file processing module might assume that a file passed to it is open and readable.

Secrets The implementation details hidden from other modules inside the module under discussion must be listed.

Error Handling Errors and exceptional conditions that may be encountered by the module must be listed, along with their causes, and the strategy used to handle them.

Uses Relationships A module uses another module if some function provided by the latter is called somewhere in the implementation of the former. Uses relationships are important because they show the modules on which each module depends for its correctness.

Implementation Details This section should discuss any security critical algorithms that your system will use. It should provide an overview of the algorithm and a description of the security guarantees that you hope to achieve by using the algorithm. Use pseudo code at your discretion, but real Java code is inappropriate. Explain how you plan to make use of cryptographic tools. (For instance, which certificates and keys, if any, are present in the system and who has access to them, how connections are securely initiated, and so on.)

Describe your protocol, including what messages are sent in response to what user action, what responses the system will take when receiving messages, and the ways in which the messages trigger changes of state.

Rationale This section explanation design decisions, listing the alternatives explored, and justifying the design decisions that were made.

Glossary As in other development documents, a glossary should be provided to minimize confusion about technical terms.

Please limit your document to no more than 5 pages. Less is fine; the more concise you can make it without giving up clarity or content, the better.

For those of you that have taken CS 162 or another class that requires a design document, note that we want you to focus on the security properties of your design. We want to be able to read your design document, and understand the security model of your system, and be convinced that you will produce a secure implementation of the protocol.

Submission Instructions

For the design review, all you have to do is send your design document to your TA via email by the deadline.

The details of the final project submission procedure will be made available at a later time.

Documentation

This section pertains to the final submission.

In addition to your design review document, your group will be responsible for providing clear documentation of your project implementation. This final document must include your design documentation (revised from the design review if necessary), implementation details, appropriate "README" information, and a security analysis explaining why your project meets the security requirements. Please use PDF, Postscript, HTML, or plain text for all documentation.

Final Design Document

This document can be reused from your original design document, or reflect changes you have made in your initial design. If you make changes, please make it clear in the final version which parts have changed and what the changes are.

Security Analysis Document

Give a convincing argument for why you think your system meets the security requirements. Explain what steps you took to ensure that only the owner of an account can access that account. Show why you think an attacker will not be able to intercept and read messages, and why an attacker would be unable to alter a message in transit.

It is not sufficient to simply state that you use encryption or the java security toolkit to solve your problems. You must explain specifically the manner in which these tools help you achieve your goals, and you should explain how you used them appropriately and correctly.

Denial of Service (DoS) Resistance

We would like you to think about how resilient your system may be in the face of a denial of service attack, though you do not have to address this in your implementation.

Assume a DoS adversary is present on the network where your instant messenger is running. This adversary may carry out attacks that include, but are not limited to, consumption or overload of system or network resources, such as, bandwidth, disk space, or CPU cycles.

You may assume that the adversary's resources are similar to those of the computer(s) the adversary is attacking, so you do not need to worry about attackers that launch distributed denial of service (DDoS) attacks from machines outside of the instant messaging network. It is assumed that the adversary has the ability to control the network and one or two clients. In your final writeup, include a section where you analyze your system in terms of its ability to withstand denial of service attacks. If your system is resistant to any DoS attacks, explain why. If your system is vulnerable to an attack, discuss how you would have done your project differently if DoS resistance had been a requirement. If you think that certain DoS attacks cannot be defended against no matter how clever you are in designing your system, explain this as well.

Source Code

Include working source code. This must contain any files necessary to run your chat system.

README

Include a README file containing the names and logins of all group members. Also, detail how to install and run the code, especially if there are any unusual steps.

Slip Days

Each group is permitted 3 slip days for the entire semester (2 projects). Once a group misses the deadline for a project (11:59 pm of the deadline day), one entire slip day will be used. Slip days cannot be partitioned.

Grading

The following is the grade breakdown for Project 1:

Design Review : 15%

Working Chat Functionality : 20%

Meeting The Security Goals : 50%

Final Documentation : 15%

Note that it will be difficult or impossible to meet the security goals if the chat functionality is not in place and working.