Development and Implementation of a Secure Instant Messenger for Public Use (November 2004)

Wesam Darwish, Wing Leung, and Megan Tiedje, *EECE 412 Students*

***Abstract*— This article describes the development and implementation of a secure Instant Messenger service using the pre-existing Microsoft Networks Instant Messenger program. Our solution involves creating a software plug-in for Trillian Pro which intercepts and encrypts messages before they are sent, and then allows the receiver to decrypt and view the original message. Also included in this article is an analysis of pre-existing solutions and an explanation of how our plug-in improves upon them.**

***Index Terms*—Eavesdropping, IM, plug-in, RSA, Trillian Pro.**

I. Introduction

INSTANT MESSENGERS (IMs) are gaining increasing popularity among regular Internet users as a cheap and effective form of communication. Unlike the telephone, IMs provide free long distance conversations and usually some sort of file transfer service; and unlike email, IM conversations occur in real-time. The drawback to IM services is that they are less secure than telephone, and often less secure than email as well. However, since IM services are free and IM conversations are usually undertaken casually, many users do not consider the security risks inherent in these systems. The users, called clients, may exchange such confidential information as passwords or bank account numbers in a way that could compromise their private information.

One prominent form of vulnerability in IM services is eavesdropping. Eavesdropping occurs when an uninvited third party reads the messages sent between communicating clients. The chances of eavesdropping increase when one or both of the following conditions are true:

- The messages have to go through a number of intermediaries,
- The messages are sent in clear text

Whether intermediaries are used to relay the IM from the sender to the recipient depends on design of the IM system. There are essentially two types of IM system designs: Client-Server type and Peer-to-Peer (P2P) type. The following figures illustrate the difference:

W. D. Author, is with the Department of Computer Engineering pursuing his master degree at the University of British Columbia.
W. L. Author, is with the Department of Computer Engineering pursuing his master degree at the University of British Columbia.
M. T. Author, is with the Department of Engineering Physic pursuing her undergraduate degree at the University of British Columbia.

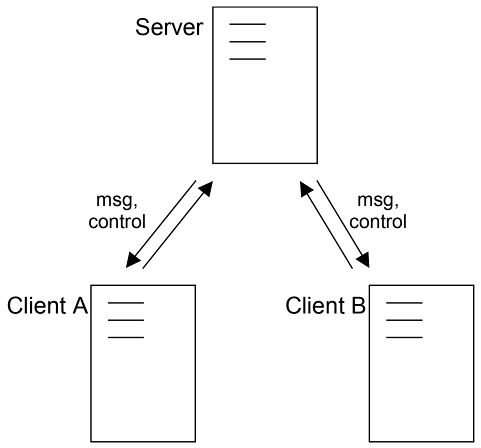


Figure 1: Client-Server Instant Messenger

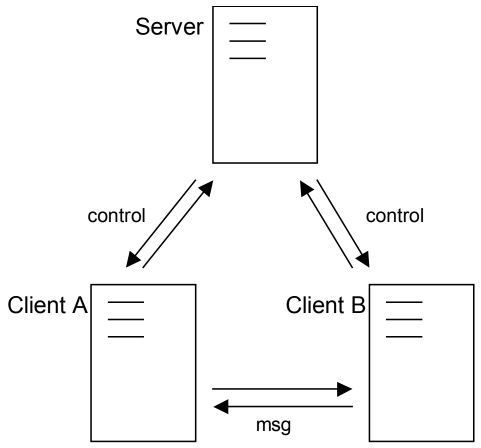


Figure 2: Peer-to-peer Instant Messenger

With a Client Server IM system, clients do not sent messages directly to each other; all message traffic is facilitated by the server. With a P2P IM system, by contrast, clients send messages to each other directly. Although it is not necessarily the case, a server system may be used for authentication, for presence services[1], and for control in both types of IM.

[1] Most IM services have some method of showing clients whether or not their particular contacts are also connected to the service at any given time. This is what is meant by "presence services."

The majority of the IM systems available are Client-Server based. This design centralizes authentication, authorization, and control services, making implementation easier. However, the Client-Server design has implications in terms of security; it could increase the number of hubs a message needs to travel through from originator to recipient, which increases the number of intermediaries the message passes through, thus increasing the chance of successful eavesdropping. With a P2P IM system, the number of hubs that a message needs to travel could decrease. For example, messages between two clients within the same domain (intranet) using a P2P IM need not travel outside of the domain. As well, eliminating the server as a route for messages eliminates the chance of secret messages being intercepted and recorded at a centralized location.

Vulnerability to eavesdropping would be substantially reduced, however, if the messages are encrypted. Eavesdropping is a problem caused by messages being "overheard", but through encryption the confidentiality of messages can be maintained regardless of whether the message is read by an uninvited party or not.

For that reason, our solution involves adding an encrypting component to pre-existing IM systems. The pre-existing factor is an important point, which touches on the design principles of psychological acceptability and ease-of-use. Since the intent of this project is to create a device that can be generally used, it is of significant benefit to base that device on a proven, popular IM system. Most users are comfortable with certain known IM systems, and might be less inclined to use a secure form of IM if it required first signing up for a new IM service, and then convincing all their friends and relatives to do the same. That is one of the ways in which our implementation improves upon some of the existing solutions.

In order to add message encryption to a pre-existing IM system, we implemented a software plug-in for Trillian Pro. Trillian is a program which incorporates a number of the most popular IM systems into a single unified user interface. Utilizing this program for our device had two basic benefits. First of all, Trillian Pro allows the creation and use of plug-ins for its system, offering the necessary interface between a user-written program and the IM services. In addition, since Trillian already accommodates a variety of popular systems, a plug-in designed for one particular system is readily extensible to other systems. Therefore, although our project focuses on implementing secure messaging for one particular service, Microsoft Networks Instant Messenger (MSN IM), we could extend the plug-in for other services.

II. Related Works

A. *Review Stage*

The following is a list of IM systems that have encryption support:

Both AOL IM and ICQ use the Blowfish Encryption algorithm, a symmetric 64-bit block cipher block which uses a variable length key. Attempts have been made to crack this cipher using differential analysis but they have not been successful.

Gaim implements public/private key encryption for both messages and file transfer. Upon starting a communication, communicating parties create a public/private key pair. Then, public keys are exchanged to implement a secured channel. This key exchange scheme is simple. However, it is weak against Man-In-The-Middle and spoofing attacks because the keys themselves are not certified. For more information, see http://gaim-encryption.sourceforge.net/.

In general, it should be noted that custom algorithms should be avoided in secure software design, particularly proprietary algorithms which are not available for analysis.

III. Solution

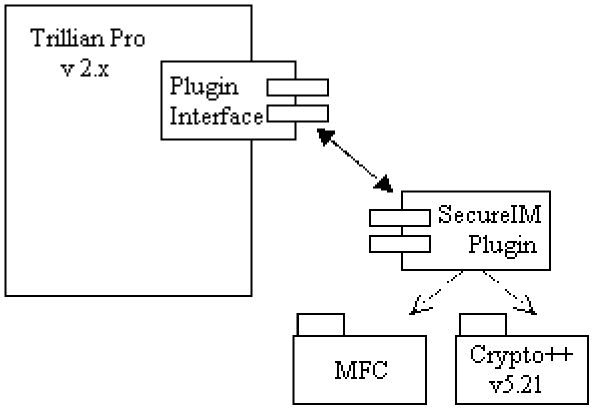
The three main factors for security are confidentiality, integrity, and availability. In this case, we assumed that availability would be within the purview of the IM system itself. Our main emphasis is on confidentiality, with some integrity functionality. Encryption is the mechanism by which confidentiality is achieved, using the Rivest, Shamir and Adleman (RSA) Public-Key Cryptography Standard (PKCS). In addition, a measure of integrity assurance is provided by our plug-in. The plug-in will respond with an error message when clear text is received or when the encrypted string fails to decrypt, which may indicate a modified message in either case.

The plug-in that we implemented is based on a threat model that assumes the IM service is used by the general public for private applications. In other words, the assumption is that public IM services are geared towards providing casual communication between friends, relatives, and co-workers. IM services are not intended for use in political, industrial, or financial applications, which is a reasonable assumption as IM services are provided free-of-charge, without client screening, and usually on an "as is" basis. Corporations and other large organizations would be expected to implement their own communication protocols for official transactions. Based on this assessment, we expect that the common threats would be due to:

- Credit card and/or bank account fraud
- Identity theft
- Any form of mischief

Our solution would not be expected to stand up to a security professional with dedicated server farms, for example, but should be expected to thwart an individual on a home computer.

A. General Architecture



The SecureIM plugin is compiled as a separate component as a Dynamic Link Library (DLL) file as required by the Trillian Pro Software Development Kit (SDK). (ref: http://www.trillian.cc/support/sdkmanual.php). The user manually loads and unloads the plugin.

The Crypto++ library used for this project is also compiled as a separate DLL. This decision was made (as opposed to linking everything with the SecureIM plugin) in order to reduce the size and the memory footprint of the plugin DLL. Furthermore, this approach allows for only the required crypto functionality to be included in the DLL.

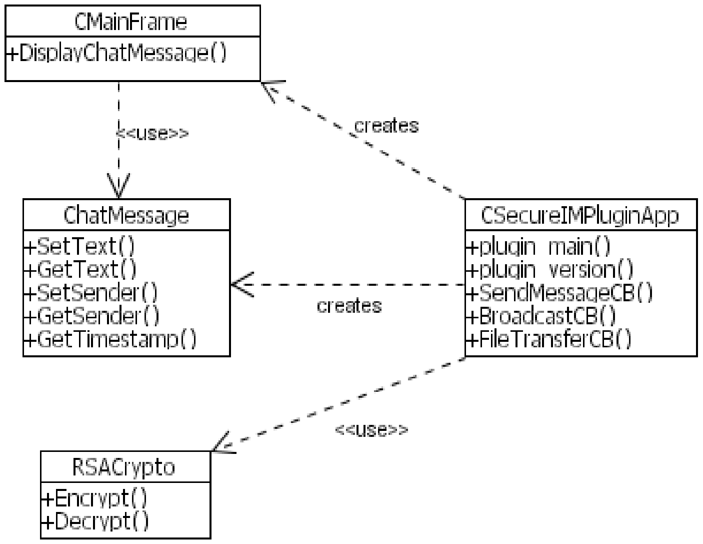
Crypto++ was used because it is a commonly used library in commercial and non-commercial products. (ref: http://www.mobiuslinks.com/links.asp?sid=1)

The SecureIM plugin uses the Microsoft Foundation Classes (MFC) for the user interface. This decision was made to minimize the development time. Other Graphical User Interface (GUI) libraries are also available, but require more effort in order to integrate them with the plugin.

B. SecureIM Plugin Architecture

First, a sent or received message is abstracted by the ChatMessage class, and the SecureIM plugin uses this class internally to pass information about the actual Trillion chat messages.

The RSACrypto class abstracts the calls into the Crypto++ library, using the RSA PKCS standard as *typedef*ed in the Crypto++ headers (RSAES_PKCS1v15). Each user is expected to have a public key already distributed to their



contacts, as well as a private key. The public key is employed by a user to encrypt a message before sending it to a second user. Upon receipt, the recipient's plug-in decrypts the message with the recipient's private key, or reports an error if the action is unsuccessful (see below for more details.)

Next, the CMainFrame class implements the SecureIM plugin GUI. This user interface displays a log window with decrypted chat messages. If the user receives a message from another party who is not using the SecureIM plug-in, or who is using the wrong public key, the log window will display a message identifying a lack of confidentiality in the received message. This GUI displays only decrypted messages; when messages are sent in clear text format the GUI displays only the error message. This feature fulfills the design principle which suggests the state of a system should be made transparent to the user.

CSecureIMPluginApp is the main class, which drives the actions performed by the plug-in, and provides entry points into the SecureIM plugin as required by the Trillian SDK (ref: http://www.trillian.cc/support/sdkmanual.php). This class also registers the following callback methods with Trillian's plug-in interface:

- SendMessageCB(): this method is called whenever the user attempts to send a message through the Trillian interface. At this point, the SecureIM plugin intercepts the message, uses the RSACrypto class to encrypt it, returns the encrypted form to Trillian for transmission, and sends the clear text version to the CMainFrame class for display.

- BroadcastCB(): this method is called whenever a message is received. Trillian sends a copy of the received message to the SecureIM plugin, which in turn attempts to decrypt it. If the message was sent in clear text format, or if it was encrypted using the wrong public key, the SecureIM plugin fails to decrypt and logs an error message as indicated in the CMainFrame class description above. The sequence diagram for this method is the complement of

SendMessageCB's.

- FileTransferCB(): this method is called to handle encryption/decryption of files in a manner similar to that used for message text.

IV. Discussion

A. Assets

We consider this to be an effective improvement to the IM system for the following reasons:

- Encryption defends against eavesdropping. As long as the RSA encryption algorithm is not broken and the keys are not compromised, the encrypted messages remain confidential. This will prevent eavesdropping attempts by an adversary who happens to intercept the message, since it is mathematically infeasible to calculate the private key from the public key and hence the message cannot be decrypted.
- The RSA encryption framework provides some defenses against spoofing. For example, if an adversary manages to gain access to another user's account and attempts to have an encrypted conversation with another user, that recipient will recognize that he or she is communicating with the wrong individual when the expected public key fails to decrypt the communication. This benefit is caused by the fact that the set of keys used for communication are not automatically linked to that account. (See the PGP keys section under Liabilities.)
- Our solution maintains the benefits of all the services provided by the IM server: authentication and control services. It is, therefore, a cost-effective solution from the perspective of hardware infrastructure and service reuse.
- We use RSA implementation from Crypto++, an open-source crypto-library. As such, the development time is minimal. All effort required goes into integration of components. Our solution is, therefore, a cost-effective solution from the perspective of software development.
- Since Trillian Pro supports multiple IM services, our encryption plug-in can be easily expanded to cover a range of systems.

B. Liabilities

Our plug-in does not address the following issues regarding IM security:

- Authentication vulnerabilities still exist. An adversary would still be able to gain access to an account if he or she managed to authenticate successfully through exercises of social engineering.
- Our solution uses the PGP services for public key exchange. We assume that key management is provided by other services and that the users know which public key to trust. This assumption may or may not hold. However, key exchange issues are not within the scope of our project.
- Our solution would probably not guard against other possible vulnerabilities introduced by Trillian Pro itself.

References

[1] Scott Van Camp, "OMG! Instant Messaging Is Becoming an Ad Vehicle" *Brandweek*, Iss. 39. vol. 45. New York, 2004, Nov 1, 2004, pp. 14. 1pgs

[2] Mike Heck, "Making Safe for the Enterprise" *InfoWorld*, Iss. 34. vol. 26. San Mateo, Aug 2004, pp. 49. 5pgs

[3] Gregg Keizer, "Public IM Threatens Entrerprise Security" *Asia Computer Weekly*, Singapore, Jul 5, 2004, pg.1

[4] Melanie Turek, "Practice safe chat" *Network World*, vol. 21, Iss 31; Framingham, Aug 2, 2004, pg.34, 2 pgs

[5] Drew Robb, "Instant Messaging" vol. 119, Iss 2; Pittsfield, pg.24, 1 pgs

[6] Juan Carlo Perez, Todd R Weiss. "Microsoft to Link Enterprise Instant Messaging Server With Rival Networks" *Computerworld*, vol. 38, Iss 29; Framingham, Jul 19, 2004, pg.10