Secure Data Exchange System : Minimizing Security Attack Risks while Preserving Bandwidth

Abdel-karim Al Tamimi **Khalid AlHokail**

aa7@cec.wustl.edu **khalid@al-hokail.com**

Abstract

In the world where Internet applications dominate data communications, a system to exchange information securely between Internet users is vital. Many solutions have been put forward to facilitate such exchange, these solutions have their own advantages and disadvantages. In this paper we introduce a thin secure layer that resides on top of the IP layer and supports encryption and compression of IP packets. Our solution provides the necessary security level to overcome most of the security risks without sacrificing performance and network bandwidth. By giving the option to choose between different levels of encryption and compression levels, the users can choose the level most suitable to their needs. In order to show the usage of our security layer, we implemented a simple chatting system that's capable of exchanging encrypted text messages and allows the clients to send encrypted and compressed files. The application also provides two ways to sniff network traffic showing the risks of exchanging information without imposing a proper security level.

Keywords:

Security Protocol, IPSec, Encryption, Rijndael, Compression, Bandwidth Consumption, Session Management, Cryptography, IP Layer, TCP/IP Suite.

See Also: IP Security : A Brief Survey Security in Wireless Data Networks : A Survey Paper

Table of Contents:

Back to Table of Contents

1. Introduction

The Internet has replaced many traditional communication systems because of its advantages in both its cost and usability. Using the Internet to share information on daily basis puts users in risk to be endangered by many Internet security attacks. Most of the data and money exchange is done these days using one of the many services provided to the users online. Such convenience comes with a high price where these communications are not always efficiently secure.

With the vast introduction of the wireless world, the exchanged information now is more prone to security risk than ever. One of the several security attacks is data sniffing, where the transmitted data is exposed to a third party and all the exchanged data is compromised. There are even commercial products that help network administrators or others to view, store and analyze exchanged data packets[EffeTech06].

The other common security attack is DoS (Denial of Service) attack, where the attacker overwhelms the victim’s host with many resource requests. A more severe version of this attack is DDoS (Distributed DoS), where the attacker uses more than one host to attack the victim’s host.

Many solutions provide the mechanism of encrypting the ongoing data exchange packets between two peers. Even when the packets are encrypted, the users are still prone to another security attack: Replay attack. Where the attacker uses pre-validated packets and sends them to one of the users to confuse and disrupt the communication.

In this project we introduce a simple to implement and easy to use infrastructure that can provide the necessary security level to exchange information between two nodes without the fear of being exposed to the sniffing attack. We also provide the necessary application level support to prevent replay attack. Because of the system's intended simplicity, it does not cover all the security risks out there. It does however, provide a base to overcome these hazards in the future.

This project report is divided into 6 sections: sections 1 and 2 cover the relevant and current solutions such as IPSec (IP Security) and SSL (Secure Socket Layer). Section 3 describes the system infrastructure, objectives, and design choices. Section 4 provides an overview of the software structure of the system. Section 5, provides a walk-through for the basic chat system introduced and the current status of the project. Finally, section 6 provides a summary and conclusion of the project.

Back to Table of Contents

2. Related Solutions

This section will illustrate two of the common solutions to facilitate exchanging data between users: IPSec and SSL. While IPSec operates on top of the IP layer, SSL operates on top of the TCP layer.

2.1 IPSec

IPSec is short for Internet Protocol Security and was developed by the IETF (Internet Engineering Task Force) to enable secure exchange of packets using the IP layer (layer 3). It is widely used in secure VPN (Virtual Private Network) communication. IPSec can work on two different encryption modes[WikiIPSec07]

1. Transport Mode

This mode only encrypts the payload (data) portion of the packet leaving the header unencrypted. This mode is mostly used in host-host communications.

2. Tunnel Mode

This mode the entire packet is encrypted and/or authenticated including the header. Which implies that another header has to be added to allow routing to work. This mode is used mainly in router-router communications.

IPSec provides two methods of securing the IP packet using one of the two protocols:

1. Authentication Header (AH)

This protocol provides integrity and data origin authentication. It can also protect against replay attacks ,repeating or delaying a previously valid packet, by using the sliding window technique. AH protects the IP header except for the mutable fields that have to change during transmission from source to destination such as the TTL field.

2. Encapsulating Security Payload (ESP)

This protocol ensures confidentiality, data origin authentication, connectionless integrity and anti-replay service. Unlike AH, ESP doesn’t protect the IP header in any way but this can be protected by using the Tunnel Mode to protect the inner IP packet but the packet header will remain unprotected.

In order for IPSec to operate properly, both the sender and receiver will have to exchange public keys. Internet Key Exchange (IKE) protocols are used to help exchange public keys between the two nodes.

2.2 SSL

SSL is short for Secure Sockets Layer which is a cryptographic protocol that provides secure communication over the internet using popular applications such as web browsers, emails and instant messaging. It was developed by Netscape Communications Corporation in 1994 and in 1999, IETF established RFC 2246 that documented Transport Layer Security (TLS) that is based on SSL. Unlike IPSec (which is implemented at the kernel level), SSL is implemented at the user level and uses TCP for reliable communication so that SSL will not have to worry about delivering the packets. It is placed above the TCP/IP layer and below the high-level application protocols. SSL provides authentication for both the client and the server. There are two methods of authentication; the first is that only the server is authenticated to ensure its identity leaving the client unauthenticated. The other method is called mutual authentication where (in addition to authenticating the server) the client is also authenticated using either his certificate or a username and a password[WikiTLS07].

SSL supports the use of various types of encryption and hashing algorithms. This is decided when the client wants to communicate with the server by sending a Client Hello message to the server with all algorithms that the client supports (along with other information such as the session ID, a random number…etc) and the server will decide which algorithms to use by selecting the strongest algorithms that both can support and then notifies the client of the choices.

Many protocols are based on SSL but the most popular protocol is HTTPS. Many websites are based on HTTPS especially the ones that accepts confidential information such as credit cards or medical records. Another popular SSL-based protocol is FTPS which is a secure FTP protocol.

In this section we described the main features of both SSL and IPSec protocols. In the next section we will discuss the system infrastructure and design choices.

Back to Table of Contents

3. System Design

In this section we discuss the system main objectives and the design choices that we had to make.

3.1 System Objectives

The system is to provide means to transfer data packets from one node to another without exposing them to security hazards, such as packet sniffing and replay attacks.

The system also needs to be extensible in the future to support other functionalities that might be added. To provide the maximum level of customizations, we allow the users to select the levels of encryption and compression best suited to their needs.

The next section will illustrate the different design choices we had to choose from and the reason behind each choice.

3.2 Design choices

In order to allow the system to be compatible with the current Internet protocols we had two options: either to implement the protocol on top of TCP (transport) layer, which is similar to SSL, or on top of IP layer, similar to IPSec. The latter option was chosen to allow us to support layer 4 protocols. This option has the drawback that it is harder to implement and the user needs to change the TCP/IP stack in order to use the system, the same problem IPSec faces. Figure 1 shows the position of the added security layer.

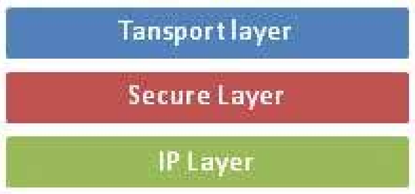


Figure 1: Secure Layer Position

In order to support the chosen layer layout, the IP packet header has to be changed before transmitting. Figure 2 below shows the layout of IP Packet.

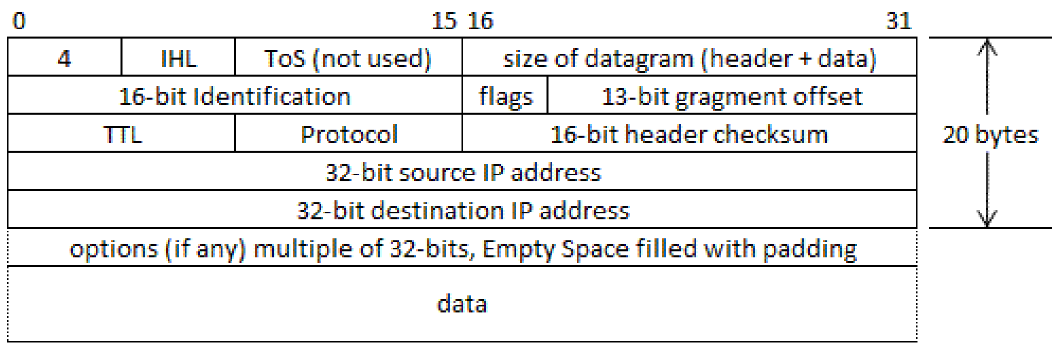


Figure 2: IP Packet Header [IPLayerRFC]

One of the options is to add the needed information in the options section, but knowing that most of the firewalls drop these packets that hold non-zero options field since it has a malicious power, we had to go for the other alternatives.

The protocol section in the IP header ,Service Access Point (SAP), which indicates the type of transport packet being carried (e.g. 1 = ICMP; 2= IGMP; 6 = TCP; 17= UDP; 255 = Reserved) is used to represent the protocol layer that we want to implement. We have chosen 255 as the number to be processed (usually used for experimental uses).

To start the encrypted session between the two points in the first time they communicate, we use public key encryption handshaking to exchange the secret key. We use RSA-1024 class implemented in .NET framework. We had to make a decision about whether we want to have a 3-way or 4-way handshaking mechanism . The selected 4-way handshaking mechanism has a drawback that it needs more overhead than the other method. But on the other hand, it supports better encryption/compression proposal when the sessions starts and on top of that it can be extended in the future to support certificate authentication like in PKINIT [PKINIT]. Figure 3 shows the basic 4-way handshaking mechanism.

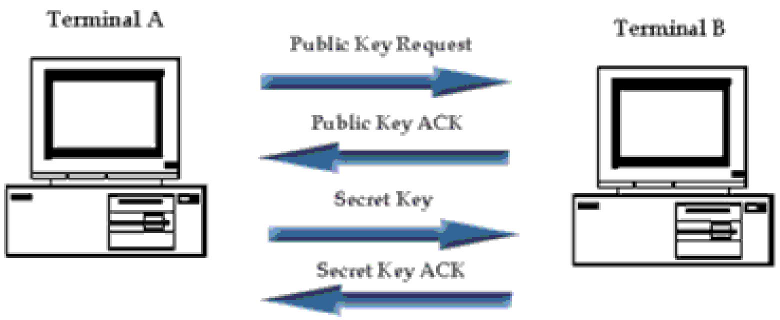


Figure 3: 4-way handshaking mechanism

For symmetric encryption, we used the AES-Rijndeal 256-CBC mode to provide strong encryption capabilities. Other encryption algorithms and modes can also be chosen in the first communication step.

A simple and efficient two-byte header design is used. Such design is small, efficient and allows further enhancements . Figure 4 shows the two byte header design.

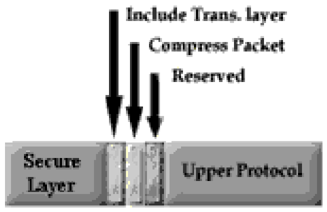


Figure 4: Secure Layer Header

The first byte contains secure layer message type field, which is 5-bits long; it holds the type of secure layer packet that is to be transmitted. Message types are public key request, public key acknowledgment (ACK), secret key, secret key ACK, end connection, end connection ACK packet, and the file transfer messages. The rest of the first byte is used for indication flags, the first one which is denoted as “Include Trans. Layer” specifies whether this packet has encrypted transport layer header or it is outside the encryption boundaries (this is another option to decrease the decryption overhead or to manage routers with some security measures). The second bit denoted as “Compress Packet” indicates if this packet is compressed or not. Compressing packet data is done before encrypting it to allow the maximum compression performance. The bit also works as an indicator to allow the post-decryption handler to decompress the received data. The last bit is reserved for future utilization. The second byte, which indicates the upper layer protocol, is an identical field to the one in the original IP header. For the time being, the upper layer protocol field is filled with number 255 to indicate that our protocol is used.

In this section we have shown the different design choices that we have chosen and the reasons behind our decisions. In the next section we will show the software architecture of the system and the technologies used to implement it.

Back to Table of Contents

4. Software Design

The implementation was done entirely using C#.NET using framework 1.1. C# is an object oriented language and it is supported in all windows operating systems after Windows 98. C# is also supported under Linux using the mono project [Mono07].

In order to implement the system design we had to use Raw Sockets, otherwise we had to change the current TCP/IP stack. The following section explains the concept behind raw socket and its usage.

4.1 Raw Socket

Raw sockets bypass the transport layer, i.e., provides a NULL transport service. Raw sockets allow reading and writing of IP datagrams in which the protocol field indicates that it is not to be processed by the kernel, e.g., routing protocols use raw sockets to exchange routing information which is not to be processed by the kernel. Raw sockets allow a user to write his\her own IP packet header. The following are the features of Raw Sockets:

- Only a user with administrative privileges (i.e. *superuser*) can create a raw socket on a UNIX system, Windows XP or Windows 2000.
- If the HeaderIncluded option is not set, then the kernel builds the IP header with the protocol in the header equal to the protocol in socket(), and the kernel starts writing the data it receives from the application just after the header.
- If the HeaderIncluded option is set, then the kernel writes the data it receives from the application starting at the first byte of the IP packet header.
- The process builds the entire IP header, except for: The IP identification field, if the application sets it to 0, the kernel always calculates and stores the IP header checksum. This checksum method is part of the implementation of the project.
- The kernel fragments raw packets that exceed the MTU.
- The kernel passes a received IP datagram to a raw socket if the following conditions are satisfied:
    1. The packet is neither a TCP nor a UDP packet.
    2. Most ICMP packets are passed to a raw socket (other than a request echo, a timestamp request and an address mask request)
    3. All IGMP packets are passed to a raw socket.
    4. All IP datagrams with an undefined protocol field are passed to a raw socket
    5. A fragmented datagram must be reassembled first before being passed to a raw socket.

To determine which raw socket the datagram is passed to, the following conditions must be satisfied:

1. A raw socket opened with a nonzero protocol field receives all datagrams with the same protocol field
2. A raw socket bound to a local IP number receives all datagrams with IP addresses matching the local IP

number

3. A raw socket connected to a foreign IP number receives datagrams originating from that remote host.

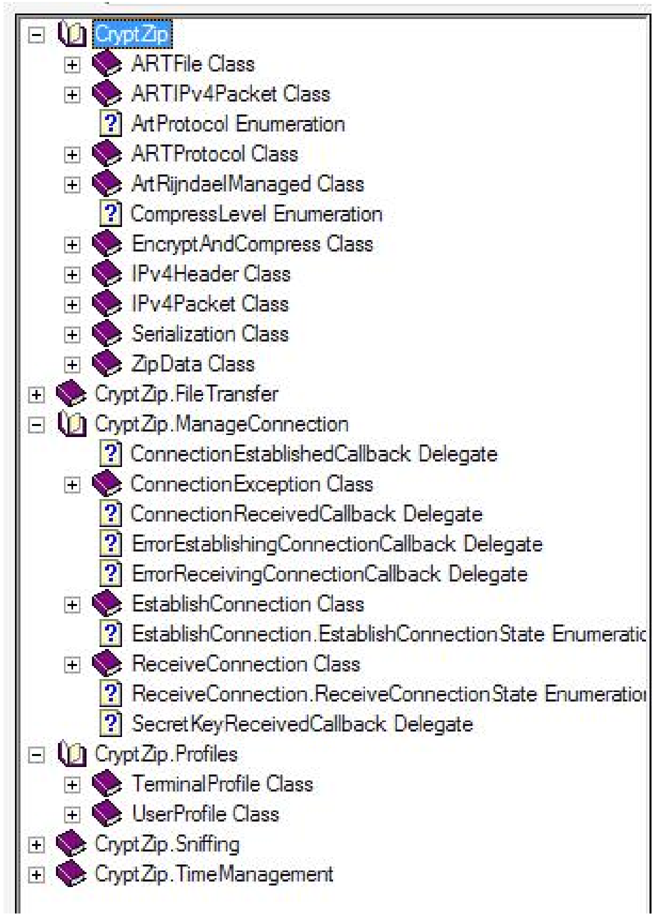
If more than one raw socket matches the above criteria, the datagram is delivered to all matching sockets.

A raw socket created with a protocol value of zero, and that is neither bound to a local IP number, nor connected to a foreign IP number receives all raw datagrams passed by the kernel to the raw sockets [WikiRawSocket].

These characteristics that RAW Sockets hold have helped us to develop the system in the way we intended. We have created all the necessary objects to support using raw socket inside the system.

4.2 CryptZip Library

In order to support both encryption and compression functionality we created library names CryptZip that provides all the necessary functionalities to implement the protocol. CryptZip acquires the functionality of SharpZipLib to support Zip compression [SharpZipLib]. Figure 5 shows the main objects implemented inside CryptZip.



```
⊟ CryptZip
    ⊞ ARTFile Class
    ⊞ ARTIPv4Packet Class
      ArtProtocol Enumeration
    ⊞ ARTProtocol Class
    ⊞ ArtRijndaelManaged Class
      CompressLevel Enumeration
    ⊞ EncryptAndCompress Class
    ⊞ IPv4Header Class
    ⊞ IPv4Packet Class
    ⊞ Serialization Class
    ⊞ ZipData Class
⊞ CryptZip.FileTransfer
⊟ CryptZip.ManageConnection
      ConnectionEstablishedCallback Delegate
    ⊞ ConnectionException Class
      ConnectionReceivedCallback Delegate
      ErrorEstablishingConnectionCallback Delegate
      ErrorReceivingConnectionCallback Delegate
    ⊞ EstablishConnection Class
      EstablishConnection.EstablishConnectionState Enumeratic
    ⊞ ReceiveConnection Class
      ReceiveConnection.ReceiveConnectionState Enumeratio
      SecretKeyReceivedCallback Delegate
⊟ CryptZip.Profiles
    ⊞ TerminalProfile Class
    ⊞ UserProfile Class
⊞ CryptZip.Sniffing
⊞ CryptZip.TimeManagement
```

Figure 5: CryptZip Library

This library supports creating and managing connections between the two communicated nodes. Also it provides the required objects to handle different user profiles. CryptZip includes all the needed functionalities to support a stand alone sniffing application through Sniffing library.

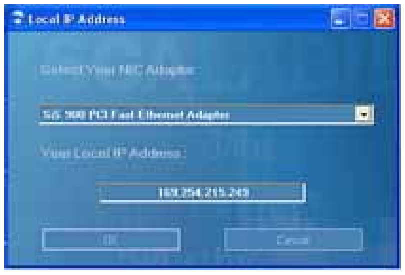
In this section we described the main software design aspects in the project design. In the next section we provide a simple walkthrough for the chat application that implements our proposed protocol.

Back to Table of Contents

5. Application Walkthrough

This section will provide a quick walkthrough for the chat application with brief description about each step.

1-NIC lookup



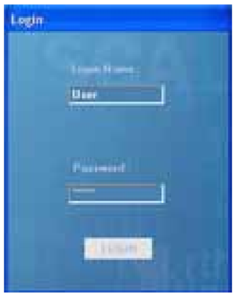
SiS 900 PCI Fast Ethernet Adapter

169.254.215.249

Figure 6: NIC Lookup

The application will detect automatically all the enabled NIC adapters and the associated IP addresses to allow the user to bind the application to a specific NIC. This function is shown in Figure 6.

2- Login Window



User

Figure 7: Login Window

The user enters his\her login name as seen in Figure 7, which is used in the chat session to identify the user. The password section not used for authentication, rather it is used as the private session key or the salt to the auto-generated session key.

3- Main Window

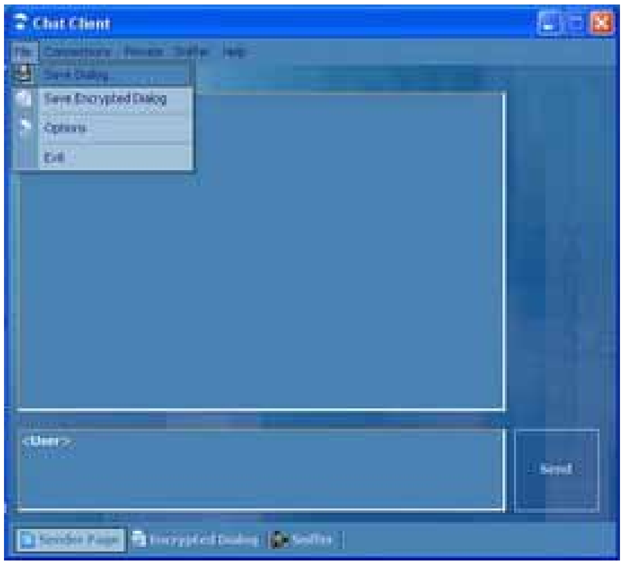


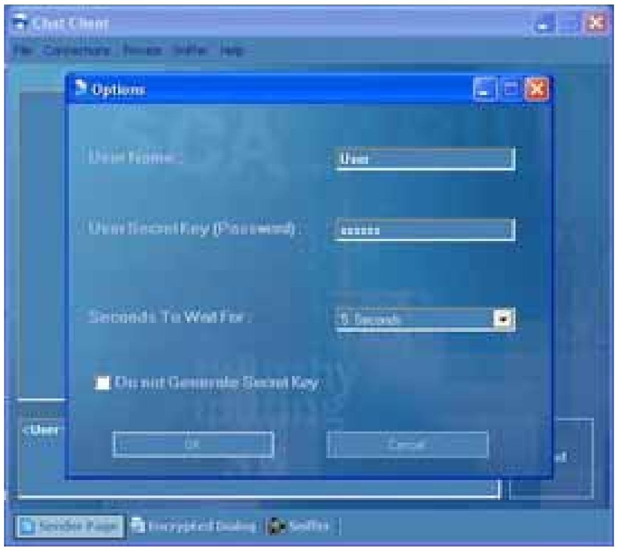
Figure 8: Main Window

Figure 8 shows the main dialog of the application, it shows the following tabs:

1. Sender Page: Chat session dialog.
2. Encrypted Dialog: The encrypted session dialog.
3. Sniffer: Show the result of the sniffer dialog.

The main window enables the user to save the chat dialog can be saved for further analysis.

4- Options Window


User Name : User

User Secret Key (Password) : ******

Seconds To Wait For : 5 Seconds

Do not Generate Secret Key

Figure 9: Options Window

Option Dialog window in Figure 9 contains the following options:

1. User Name: which can be changed without the need to restart the entire application.
2. User Secret Key: which also can be changed withoug the need to restart the entire application.
3. Seconds to wait For: which is the maximum amount of time to wait between each message, if the expected

message did not come during this time the connection will be dropped. This feature helps against DoS attacks.

4. Do not generate Secret Key: Secret key is generated and changed every 5 minutes (this option can be changed with different user profiles). User can use this option to boost performance at the risk of being a target to replay attack. The default value is false.

5- Managing Connections

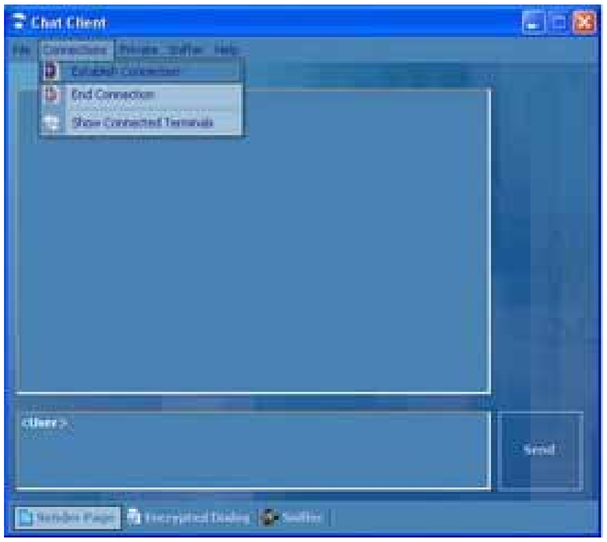


Figure 10: Managing Connections

To establish a new connection, go to connections menu (as seen in Figure 10) and choose Establish connection command. This will show the following window in Figure 11:

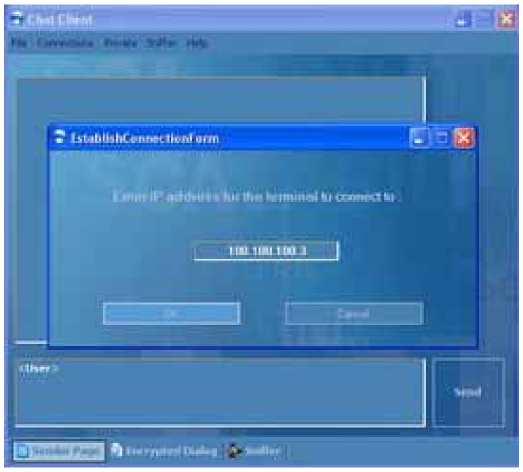
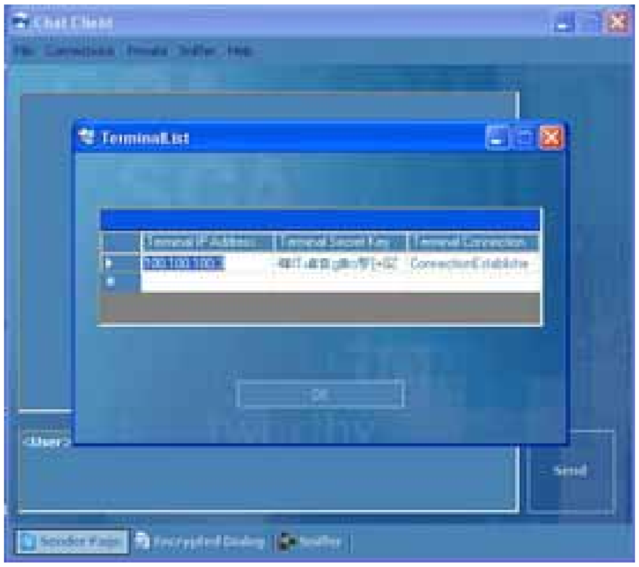


Figure 11: Establish New Connection

In this window you can choose the IP Address you want to connect to. Once the connection is established you can view the connected terminals by going to Connection menu then click on Show Connected Terminals as seen in Figure 12.



Terminal IP Address	Terminal Secret Key	Terminal Connection
100.100.100.3	[illegible]	ConnectionEstablishe

Figure 12: List of Current Terminals

6- Private Message and File Sharing

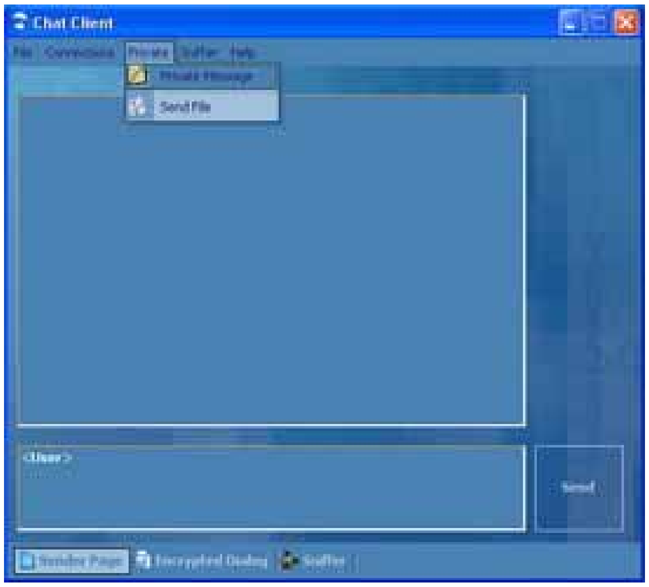
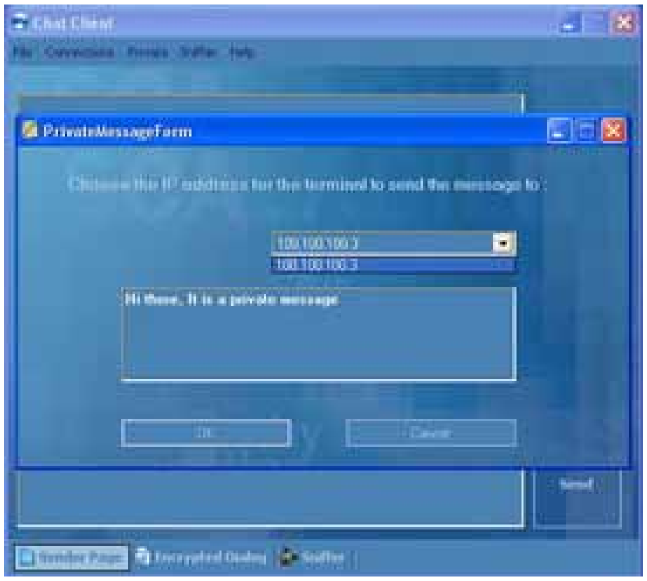


Figure 13: Private Message and File Sharing

When you are connected to more than one terminal at a time, you may want to send a private message to a certain user; in addition to that you may want to share a file with that user. Private menu shown in Figure 13 allows you to do both functionalities. Figure 14 below shows how you can send a private message:



Choose the IP address for the terminal to send the message to :

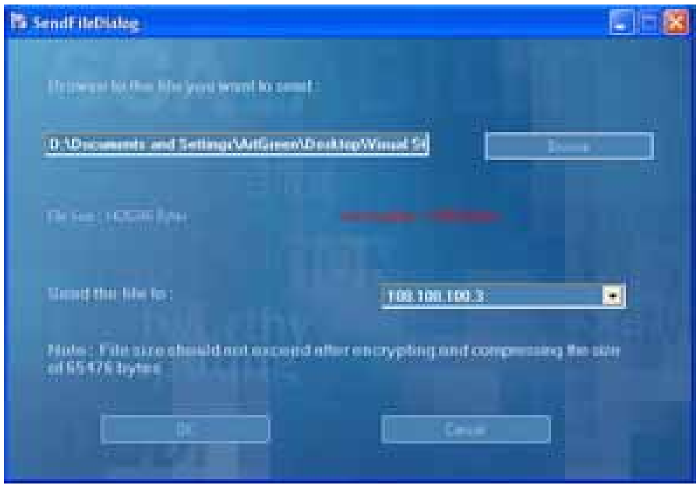
100.100.100.3

100.100.100.3

Hi there, It is a private message

Figure 14: Sending a Private Message

Sharing a file is a very easy process. The default behavior when sending a file is to compress and encrypt the file before sending it. Figure 15 below shows the difference between the size of the transmitted file before and after applying encryption and compression. (1426246 bytes to 118000 after compression and encryption, almost 8.5% of the old size is transmitted). This improvement of bandwidth consumption comes on the expenses of increased processing power needed to send the packet. But the new machines are capable of doing such processing without showing any slowing in the transmission time.



Browse to the file you want to send :

D:\Documents and Settings\AtGreen\Desktop\Visual St

File Size : 1426246 Bytes [illegible]

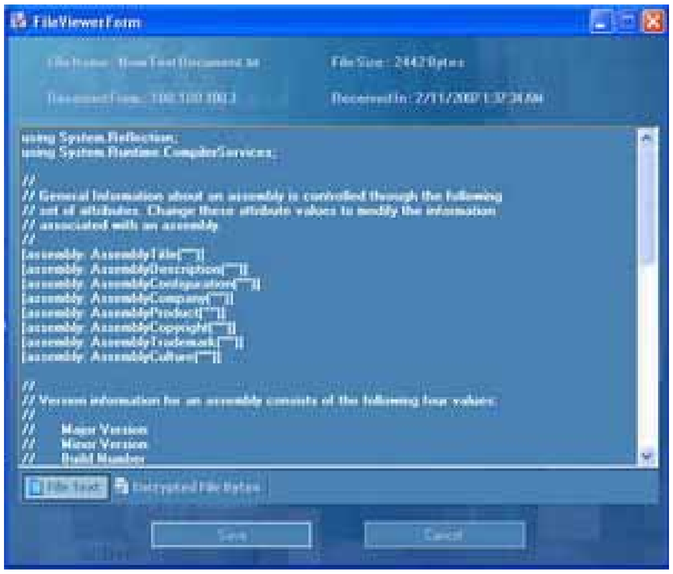
Send the file to :

100.100.100.3

Note : File size should not exceed after encrypting and compressing the size of 65476 bytes

Figure 15: Sending a File Window

Figure 16 shows how the file is received on the other end after decryption and decompression.

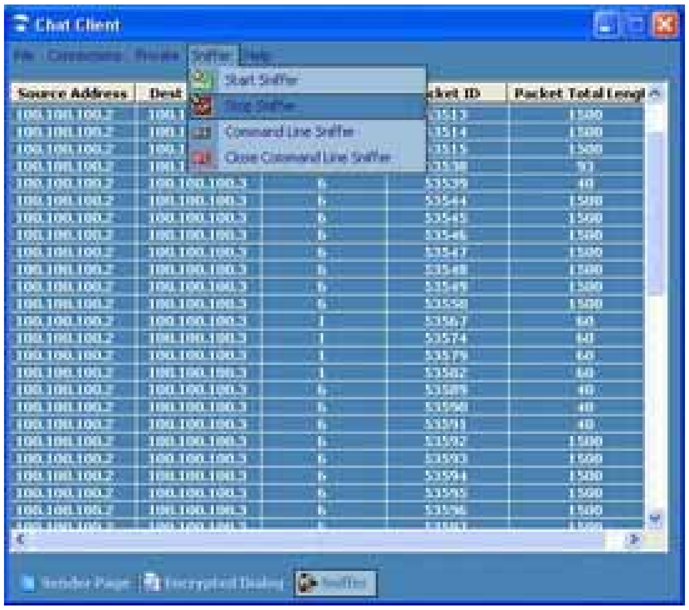

FileViewerForm
File Name : New Text Document.txt
File Size : 2442 Bytes
Document From : 100.100.100.3
Received In : 2/11/2007 1:37:34 AM
File Text
Encrypted File Bytes
Save
Cancel

```
using System.Reflection;
using System.Runtime.CompilerServices;

//
// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
//
[assembly: AssemblyTitle("")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("")]
[assembly: AssemblyCopyright("")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

//
// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
```

Figure 16: Viewing the Received File

7- Sniffer

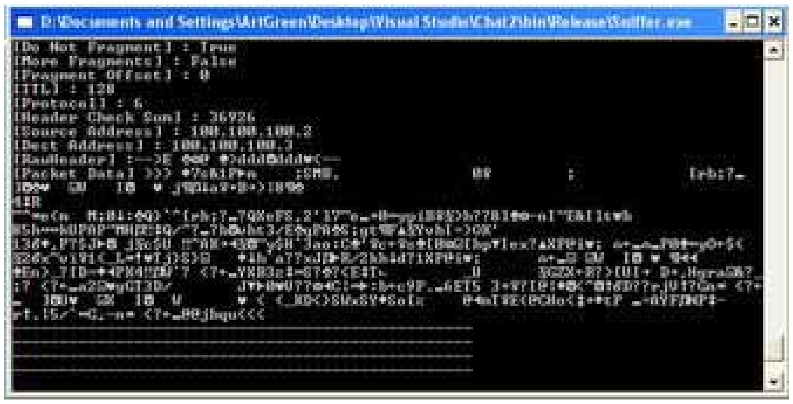
Sniffer implementation is divided into two sections, the simple sniffer and the command line sniffer. The simple sniffer shows only 5 fields of the IP header captured using the sniffer thread: Source Address, Destination Address, Protocol, Length and Packet ID. Command line sniffer shows all the information in the IP header along with the packet contents. The simple sniffer is shown in Figure 17.



Source Address	Dest		cket ID	Packet Total Lengt
100.100.100.2	100.1		53513	1500
100.100.100.2	100.1		53514	1500
100.100.100.2	100.1		53515	1500
100.100.100.2	100.1		53538	93
100.100.100.2	100.100.100.3	6	53539	40
100.100.100.2	100.100.100.3	6	53544	1500
100.100.100.2	100.100.100.3	6	53545	1500
100.100.100.2	100.100.100.3	6	53546	1500
100.100.100.2	100.100.100.3	6	53547	1500
100.100.100.2	100.100.100.3	6	53548	1500
100.100.100.2	100.100.100.3	6	53549	1500
100.100.100.2	100.100.100.3	6	53550	1500
100.100.100.2	100.100.100.3	1	53567	60
100.100.100.2	100.100.100.3	1	53574	60
100.100.100.2	100.100.100.3	1	53579	60
100.100.100.2	100.100.100.3	1	53582	60
100.100.100.2	100.100.100.3	6	53589	40
100.100.100.2	100.100.100.3	6	53590	40
100.100.100.2	100.100.100.3	6	53591	40
100.100.100.2	100.100.100.3	6	53592	1500
100.100.100.2	100.100.100.3	6	53593	1500
100.100.100.2	100.100.100.3	6	53594	1500
100.100.100.2	100.100.100.3	6	53595	1500
100.100.100.2	100.100.100.3	6	53596	1500

Figure 17: Simple Sniffer Window

The command line sniffer is a separate process that can be called without using the chat application. This is shown in Figure 18.

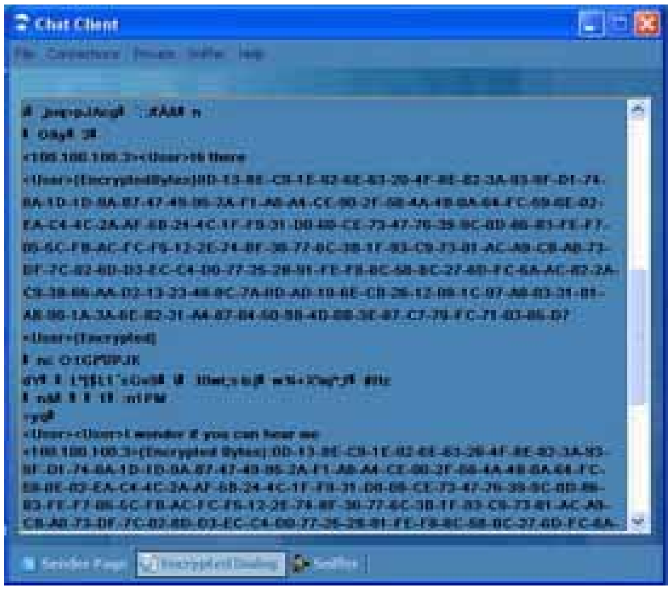


```
[Do Not Fragment] : True
[More Fragments] : False
[Fragment Offset] : 0
[TTL] : 128
[Protocol] : 6
[Header Check Sum] : 36926
[Source Address] : 100.100.100.2
[Dest Address] : 100.100.100.3
[RawHeader] :--->E [illegible]
[Packet Data] >>> [illegible]
[illegible]
```

Figure 18: Command Line Sniffer

8- Chat Window and Encrypted Dialog

When using the chat system, the user can view the transmitted encrypted information. Figure below shows the dialog window of encrypted chat dialog. Notice that the window contains both plain text message and the encrypted output.



```
[illegible]
<100.100.100.3><User>Hi there
<User>[EncryptedBytes]0D-13-8E-C9-1E-82-6E-63-20-4F-8E-82-3A-83-8F-D1-74-
8A-1D-1D-0A-87-47-45-05-2A-F1-A8-A4-CC-00-2F-08-4A-48-0A-64-FC-69-0E-02-
EA-C4-4C-2A-AF-5B-24-4C-1F-F9-31-D0-80-CE-73-47-76-39-9C-0D-86-83-FE-F7-
[illegible]
<User>[Encrypted]
[illegible]
<User><User>I wonder if you can hear me
<100.100.100.3>[Encrypted Bytes]:0D-13-8E-C9-1E-82-6E-63-20-4F-8E-82-3A-83-
[illegible]
```

Figure 19: Encrypted Chat

As shown in the screenshot in Figure 19, the attacker using any of the sniffing tools, cannot read the contents of the transmitted messages. And because of the password regeneration feature, any offline attack is useless unless the user chooses not to use this feature. Compression has helped limiting the bandwidth consumption that usually accompanies encryption techniques and the associated padding. Though using compression can lead to performance overload, today’s machines are capable of such intensive load. Compression also added extra protection to the transmitted data, since the compressed data is not comprehensible.

This section illustrated the basic usage of the chat system application. The project still need further development to support larger file types and give the users more options to change from the options menu.

Back to Table of Contents

6. Summary

In this paper we introduced our solution to provide a simple infrastructure to exchange data between two nodes. Our solution considers the current technologies used nowadays. Our main goal is to provide a simple structure for a protocol to allow further modifications and alterations. Our proposal is not intended to be secure against all security attacks at this level, but it manages to solve some of the common security risks. Adding compression to the encryption process adds both overhead and enhances the security level of the transmitted data. Users have the options to change the default profile of the compression and encryption process, where they can meet their specific requirements.

Back to Table of Contents

7. References

1- [IPLayerRFC] INTERNET PROTOCOL [Online] http://www.faqs.org/rfcs/rfc791.html.

2- [WikiIPSec] Wikipedia. 2007. IPSec. *Wikipedia.* [Online] November 11, 2007. [Cited: November 20, 2007.] http://en.wikipedia.org/wiki/IPsec.

3- [WikiTLS] Wikipedia. 2007. Transport Layer Security. *Wikipedia.* [Online] November 16, 2007. [Cited: November 20, 2007.] http://en.wikipedia.org/wiki/Transport_Layer_Security.

4- [WikiRawSocket] Wikipedia. 2007. Raw Sockets. *Wikipedia.* [Online] November 16, 2007. [Cited: November 20, 2007.] http://en.wikipedia.org/wiki/Raw_socket.

5- [Mono07] Mono. 2007. Mono. *Mono.* [Online] 2007. [Cited: November 20, 2007.] http://www.mono-project.com/Main_Page.

6- [EffeTech06] EffeTech. 2006. Packet Sniffer, HTTP Sniffer, Password Sniffer, MSN Sniffer - EffeTech. *Packet Sniffer, HTTP Sniffer, Password Sniffer, MSN Sniffer - EffeTech.* [Online] September 3, 2006. [Cited: November 23, 2007.] http://www.effetech.com/.

7- [SharpZipLib] IC# Code. 2007. *The Zip, GZip, BZip2 and Tar Implementation For .NET* [Online] [Cited: November 23, 2007.] http://www.icsharpcode.net/OpenSource/SharpZipLib.

8- [PKINIT] PKINIT. *Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)* [Online] [Cited: November 23, 2007.] http://www.ietf.org/rfc/rfc4556.txt.

9- [Jain07] Jain, Raj. 2007. IP Headers. *Raj Jain.* [Online] November 20, 2007. [Cited: November 20, 2007.] http://www.cse.wustl.edu/~jain/cse571-07/ftp/l_13ips/sld014.htm.

Back to Table of Contents

8. Appendix A: Abbreviations

ACK	Acknowledgment
AES	Advanced Encryption Standard
AH	Authentication Header

CBC	Cipher Block Chaining
DDoS	Distributed Denial of Service
DoS	Denial of Service
ESP	Encapsulating Security Payload
FTP	File Transfer Protocol
FTPS	Secure File Transfer Protocol usign SSL
HTTPS	Secure Hyper Text Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
IHL	IP Header Length
IKE	Internet Key Exchange
IP	Internet Protocol
IPSec	Internet Protocol Security
MTU	Maximum Transmission Unit
NIC	Network Interface Card
RSA	Named after its creators: Ron Rivest, Adi Shamir, and Leonard Adleman
SFTP	SSH File Transfer Protocol
SSH	Secure Shell

SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
ToS	Type of Service
TTL	Time To Live
UDP	User Datagram Protocol
VPN	Virtual Private Network

Back to Table of Contents

Date Last Modified: Friday, November 23, 2007

Note: This paper is available online at http://www.cse.wustl.edu/~jain/cse571-07/ftp/XXX/index.html