Steganography Project

**Introduction**

The purpose of the project is to send secret data over the wire employing Skype’s VOIP packets. The project comes with a simple UI on top of Skype API which helps the user to call a friend and one’s the connection is established between the two, it allows the users to share the secret data through our application without the Skype realizing it. Hence, the secret chat will not be stored in Skype application because it never happened on Skype.

This is a convenient means of exchanging messages, which requires mandatory use of our application by both the parties to successful exchange of secret data. Our application is a java based desktop application and its UI is developed in Swing.

**System Requirements**

This application can be run on any system which has the following installed:

* Java 1.4 and above
* WinPcap (for windows)
* Skype

**Operation**

Start Skype and sign into it, then double click on our application and start it. A UI pops up and it shows the user’s Skype friend list. Select a friend and press the Call button which will initiate a call to the friend. Once the call is started, user can type the secret message in the text area and press the Send button. The friend at the other end will receive the message in a similar chat window.

**Functional Details**

The project is functionally divided into two modules which are listed below:

**Message Sender**

This module is responsible to accept a message string from the chat window, break down the message into several chunks, embed individual chunk into the VOIP packets and then send it over the wire, during the Skype call. After sending the chunk of packet, it awaits acknowledgement and upon receipt of acknowledgement, it initiates transfer of another chunk and the process goes on till the chat completes.

**Message Receiver**

This module is responsible to listen to the network card, intercepts packets, pick up the packets and filter out the required packets. After filtering the packets, it extracts the secret data and assembles the chunks in proper order, based on the sequence number of the packets and builds the complete message. After receiving a complete group/chunk of data, it sends out a receipt acknowledgement. The process continues till the end of the communication.

**Technical Details**

The class is designed to be a singleton class, which ensures that on one instance of this application running on a given system, everything is performed by a single instance of this class.

The class is an entry point and manages all the invocations between the UI  class **{MainFrame}**, the network packet capturer class **{Capturer}**,  the packet transporter class **{Transport}**, the message sender class **{MsgSender}**, the message receiver class **{ MsgReceiver}** and the class **{SkypeControl}** which serves as a wrapper over the Skype API to make the Skype related calls easier.

The class also maintains a history of all the messages being received or sent.  Also, all the methods of this class are directly called from the UI.

**What happens when the user types a message string and the Send button is pressed?**

When the send button is pressed, the **sendMsg(String msgStr)** method of the **{Manager}** class is invoked. The following steps are performed in sequence as a part of this call:

* The message string is added to the message history.
* The message string is added to the **{MsgSender}** message queue. A lock is acquired on this queue before adding this message.
* Notifies all the modules are waiting to process the message which should be sent to remove the lock.
* A {**MesssageDispatcher}** thread is always waiting to acquire a lock on the message queue of the **{MsgSender}.**
* Upon receiving a notification for the release of lock on the queue, it polls the queue, if the queue is empty; it waits on the queue again.
* If the queue contains a message, it starts the process to establish the three way handshake with the application on the other hand and starts the process to send a Start Message Control Packet. The process involves calculating the number of groups which will be sent as a part of sending the message string.
* The number of groups is simply calculated by dividing the length of the message string by 10. For more reference see MsgTransfer. calcNumGroups(String msgStr).
* The application maintains a TransferStatusMachine to mark each of the status of the message transfer which will remain at CLOSED state in the beginning.
* The message string is sent to the **{Transport}** class which maintains a queue of outgoing network packets.  For more reference please see MsgTransfer. sendMsgStartPacket(int numGroups).
* If there is no packet present in the queue, sleep for 20 milliseconds and try polling again.
* If there is a packet present in the queue, build a control packet of type {RtpStartMsgPacket} with empty byte array of 100 bytes and embed the number of groups in the timestamp. For more reference see RtpStartMsgPacket.packet(int numGroups)
* Set the sequence number of this packet as zero, which is the SYN packet.
* Send this packet over the data gram socket and store the information of this packet into a file outPacketData.txt
* After sending the start packet wait for acknowledgement MsgTransfer.waitToReceiveAck(int waitTime) .
* The RtpMsnTransPacketHandler handles all the incoming messages received by the Capturer. It checks if the received packet is a valid ACK packet then the waitToReceiveAck method returns the ACK packet, in case the ACK packet is not received then it sleeps for the waitTime.
* Once a valid ACK packet is not received, the application repeats the process of sending a Start Message Control Packet until the process results in receipt of a valid ACK packet.
* If a valid ACK packet is received, then the TransferStatusMachine status to START.
* Once this is done, then the application is ready to send actual message in a secret way. With the already calculated number of groups as N, it tries to send the whole message in N groups.
* It divides the total message into N groups each of length 10 and the last group of length equal to remaining characters.’
* Upon successful division, every group has an index, and the application polls the outgoing packet from the Transporter queue of outgoing packets. Once a outgoing packet is present, it picks one byte of the data in sequence from the current group, sets it as a payload immediately after the header length
* Once this packet is successfully rebuilt, it sends the packet over the data gram socket.
* The RTP packet information is stored in the outPacketData.txt.
* Once the complete group is sent, it sends a control packet over the data gram socket with the group index information and sequence number zero, in a way similar to sending the Start Message Control Packet. For more information see MsgTransfer.sendCtlPacket(int index).
* After sending the control packet wait to receive a valid acknowledgement. If a valid acknowledgement is not received, it tries to send the control packet again and waits until a valid acknowledgement is received.
* In case a valid acknowledgement is received, the TransferStatusMachine status is changed to START\_SUBDATA
* In a similar way it keeps sending the data in each group and if it fails anywhere in the middle, it ends the complete message to the end of the message queue to be sent later.
* If all the groups were transferred properly, it changes the status of the TransferStatusMachine to CLOSED which means that the given message was successfully transferred.

**What happens at the receiving end?**

* At the receiving end the Capturer is consistently capturing the packets arriving on the network interfaces.
* For each incoming packet, the RtpMsnRecPacketHandler handler checks if it is a start packet, a check packet or a data packet.
* A start packet is identified by the start sequence number which is zero and the cmd tag embedded in time stamp which is one.  For more information please see RtpStartMsgPacket. isStartPacket(**long** timeStamp, **int** l).
* ***In case the packet  is a start message packet,*** it extracts the number of groups information and allocates an array of maximum size equivalent to the product of (number of groups , group size and valid RTP size i.e. one)
* This array will be used to store the complete message string which will be transferred as a part of a single message.
* After extracting all these information, the receiver sends an acknowledgement packet which has the same data (the number of groups it is expecting) and changes the status of the TransferStatusMachine of the receiver to START.
* ***In case the packet is a data packet***, a data packet is identified by its sequence number which is greater than zero and lesser than the group size (in our case ten).
* Upon receipt of a valid data packet, extract its sequence number and extract the one byte which is immediately after the header of the packet.
* Store this byte of data in the array which was allocated at the time it received the Start Message Control Packet at the index groupIndex \* groupSize + sequenceNumber -1.
* Keep doing this for all the packets of the same group and once the group is fully received, change the Receiver’s status of the TransferStatusmachine to  START\_SUBDATA.
* After this, it receives the check packet which is just to verify if the group was successfully received.
* It sends a valid ACK packet once the group is successfully received.
* The same process is repeated till all the groups are successfully received.

**An example of run of the application**

**Message to be sent:** **abcdefghijklmnopqrstuvwxyz**

**Steps involved in sending:**

1. Calculate the number of groups = [message length / 10] = 3
2. Sending Start Message Control Packet:
   1. Create a RTPControlPacket and assign a RTPRawPacket of packet length 100 and a packet represented by a buffer array of 100 bytes. Change the state of the TransferStatusMachine to INIT
   2. Sets the payload type to 0x0F and time stamp to a random long value.
   3. Sets the sequence number of the packet to 0x00.
   4. sets the time stamp of the packet to a long value calculated based on the number of groups by the following procedure:

* Assuming the number of groups is 3 (00000011)
* Left Shift the number 0x03 by 2 bits to get 00001100
* OR it with 00000001 to get 00001101 as cmdLong. Use cmdLong value with the random time stamp of the packet to generate a new long value and set the value as the new time stamp. e.g. calculation  newTimeStamp = (timeStamp & (long)0xFFFFFF00) | cmdLong

1. Set the sequence number of the packet to zero.
2. Get the UDP packet from the RTP packet and in turn get the Datagram packet form the UDP packet and send it on the Datagram socket.
3. Wait and scan the incoming packets and check if the incoming packet has a sequence number zero and a tag calculated using the times tamp of the packet and the following formula byte tag = (byte)(timeStamp & (long)0x00000003).
4. This is the same thing we did in step d. If the tag matches 3(00000011) and sequence number is zero then it is the acknowledgement of the control packet we just sent.
5. Change the state of the TransferStatusMachine to START after receiving the ACK packet.
6. Start sending the data. This is done in the following steps:
7. Divide the data into the number of groups, hence we will end up with three groups as follow:

* Group 1: abcdefghij
* Group 2: klmnopqrst
* Group 3: uvwxyz

1. Send each group at a time, pull out an outgoing packet and create an RTP data packet of 100 byte length and a random sequence number and random time stamp and packet type 0x0F.
2. Set the sequence number of this packet to 1, set the time stamp of the time stamp to a long value equivalent to 0xFFFFFFFF.
3. Add the first byte of the Group 1, i.e. a at the end of the header of the packet which is data[header\_length + 1]. The packet length becomes header\_length + 1.
4. Get the UDP packet from the RTP packet and in turn get the Datagram packet form the UDP packet and send it on the Datagram socket.
5. Repeat the steps 9 more times to send the complete group (b, c, d, e, f, g, h, i, j) and keep changing the sequence number one through ten.
6. After sending 10 packets (which is equivalent to the Group 1), send a check packet with sequence number zero and send the group number embedded in the packet's time stamp property.
7. Wait to receive the acknowledgement for the check packet; if an ACK is received then change the state of the TransferStatusMachine to START\_SUBDATA and continue sending the group 2, the sequence number starts again from one to ten for the group 2.
8. Send another Check packet after sending the ten packets containing one byte each from k, l , m , n , o , p , q, r, s, t.
9. Wait to receive the acknowledgement for the check packet, if an ACK is received then continue sending the group 3, the sequence number starts again from one to six for the group 3.
10. Now receive the next ACK packet and then change the status of the TransferStatusMachine to CLOSED.

**Steps involved in receiving:**

1. Check the entire incoming packets, and find if the packet is a Start Packet. The packet is identified by its sequence number and its time stamp. If the sequence number is zero and the tag calculated by the following tag = (byte)(timeStamp & (long)0x00000003) is 0x03, it is a start message control packet.
2. Upon receipt of the Start Message Control Packet, extract the number of groups to be received and allocate a byte array DATA of length equivalent to 10 \* number of groups, this will be used to rebuild the complete message again.
3. Send out an ACK for the Start Message COntrol Packet.
4. Receive the Incoming packets and check if the packet is a valid data packet. A valid data packet has a sequence number from one to ten and a time stamp whose tag calculated by the following tag = (byte)(time & (long)0x000000FF), will return 1.
5. If the packet is a data packet, extract one byte right after the header\_length and store it at DATA[(groupIndex \* 10) + sequenceNumber - 1], where 10 is the size of the group. The groupIndex is zero in the beginning.
6. Repeat this for all the data packets we receive place it in the DATA array at appropriate location.
7. Upon receipt of a check packet, increase the number of the groupIndex by 1 and send an ACK for the check packet.
8. Repeat the above steps till we receive the data packets and check packet.
9. The DATA array in the end will have the content abcdefghijklmnopqrstuvwxyz.