Tutoral 2 -- SocketPro secure communication and built-in bi-directional message pushes

Contents:

1. Introduction

In short, we need to secure sensitive data movement among computers such as passwords, credit cards, personal identification numbers, financial data, and so on. To achieve this purpose, SocketPro uses popular openssl (Linux) and SSPI (windows) libraries across all supported platforms.

A typical distributed application is involved with a network communication between the client and server only, but there is no communication among connected clients at all. However, this typical architecture does not meet application requirements under many cases. An application may require some ways to enable communications among clients whenever an interested thing happens. This type of communication is similar to Internet chat among a group of people. Therefore, SocketPro provides this type of chat service, which is a built-in service for all of connected socket connections.

For your information, the built-in chat service is also called message bus, message push, subscribe/publish communication model or pattern, or real-time notification service. These terminologies are used interchangeably within SocketPro.

Finally, the tutorial sample projects are located at the directory ..\tutorials\(csharp|vbnet|cplusplus|java\src|ce|python)\pub_sub

2. Use SSL/TLS for secure communication

To use SSL/TLS for secure communication, you must enable it at both client and server sides.

Enable SSL/TLS at server side: It is very simple to enable SSL/TLS at server side. All you need to do is call the method *UseSSL* as shown in the below Figure 1 before running a SocketPro server.

```
23      static void Main(string[] args) {
24          CMySocketProServer MySocketProServer = new CMySocketProServer();
25
26          //test certificate and private key files are located at ../SocketProRoot/bin
27          if (System.Environment.OSVersion.Platform == PlatformID.Unix)
28              MySocketProServer.UseSSL("intermediate.cert.pem", "intermediate.key.pem", "mypassword");
29          else {
30              MySocketProServer.UseSSL("intermediate.pfx", "", "mypassword");
31
32              //load cert and private key from windows system cert store
33              //MySocketProServer.UseSSL("root"/*"my"*/, "UDAParts Intermediate CA", "");
34          }
35          if (!MySocketProServer.Run(20901))
36              Console.WriteLine("Error code = " + CSocketProServer.LastSocketError.ToString());
37          Console.WriteLine("Input a line to close the application ......");
38          Console.ReadLine();
39          MySocketProServer.StopSocketProServer(); //or MySocketProServer.Dispose();
40      }
```

Figure 1: Enable SSL/TLS at server side as shown at the line 28 (Linux) or 30 (windows)

This sample uses sample CA root cert (ca.cert.pem which will be used at client side), server certificate (intermediate.cert.pem) and private key (intermediate.key.pem). The third input ("mypassword") is password to be used to read a password-protected private key from the private key file intermediate.key.pem. All of them can be found at the directory ../SocketProRoot/bin.

Enable SSL/TLS for client side and server certification authentication: To enable SSL/TLS at client side, we need to set it as you see the TLSv1 at the end of line 29 on a connection context as shown in the following Figure 2.

```
27      static void Main(string[] args) {
28          Console.WriteLine("Input your user id ......");
29          CConnectionContext cc = new CConnectionContext("localhost", 20901, Console.ReadLine(), "MyPassword", tagE
30
31          //CA file is located at the directory ..\SocketProRoot\bin
32          CClientSocket.SSL.SetVerifyLocation("ca.cert.pem");
33
34          //for windows platforms, you can also use windows system store instead
35          //CClientSocket.SSL.SetVerifyLocation("my"); //or "root", "my@currentuser", "root@localmachine"
36
37          using (CSocketPool<HelloWorld> spHw = new CSocketPool<HelloWorld>()) //true -- automatic reconnecting
38          {
39              spHw.DoSslServerAuthentication += (sender, cs) => {
40                  int errCode;
41                  IUcert cert = cs.UCert;
42                  Console.WriteLine(cert.SessionInfo);
43                  string res = cert.Verify(out errCode);
44                  //do ssl server certificate authentication here
45                  return (errCode == 0); //true -- user id and password will be sent to server
46              };
47
48              //error handling ignored for code clarity
49              bool ok = spHw.StartSocketPool(cc, 1, 1);
50              HelloWorld hw = spHw.Seek(); //or HelloWorld hw = spHw.Lock();
```

Figure 2: Enable SSL/TLS at client side and certification authentication

In addition to enable SSL/TLS at client side, it is also required to authenticate a server certificate as expected before sending any sensitive data from the client to a remote server. This step ensures there is no possibility of a middle-man attack.

To do so, we need to set a callback and check if a certificate from a server is expected. SocketPro uses an event as shown from lines 39 to 46 of the above Figure 2. Inside the callback, we are able to get an interface to the certificate from the remote server. Afterward, we can simply check it with a local saved certificate as you can see at line 43. After both match, certificate chain is verified successfully and the callback returns true, SocketPro will send sensitive data such as user id and password. Otherwise if it returns false, SocketPro does not send the two sensitive data but closes socket connection automatically.

3. SocketPro built-in bidirectional message pushes

SocketPro supports two types of real-time notification, one-to-many and one-to-one.

One-to-many: As shown in Figure 3, a sender is able to send a message to one or more groups of chatters or subscribers. You can do so from either the client or server side. SocketPro server is able to distribute the message based on an array of input chat group identification numbers (or topic ids) on behalf of the sender. If you use this notification model, a listener or subscriber must first subscribe to one or more chat groups for expected groups of messages. However, a message publisher does not have to do so.

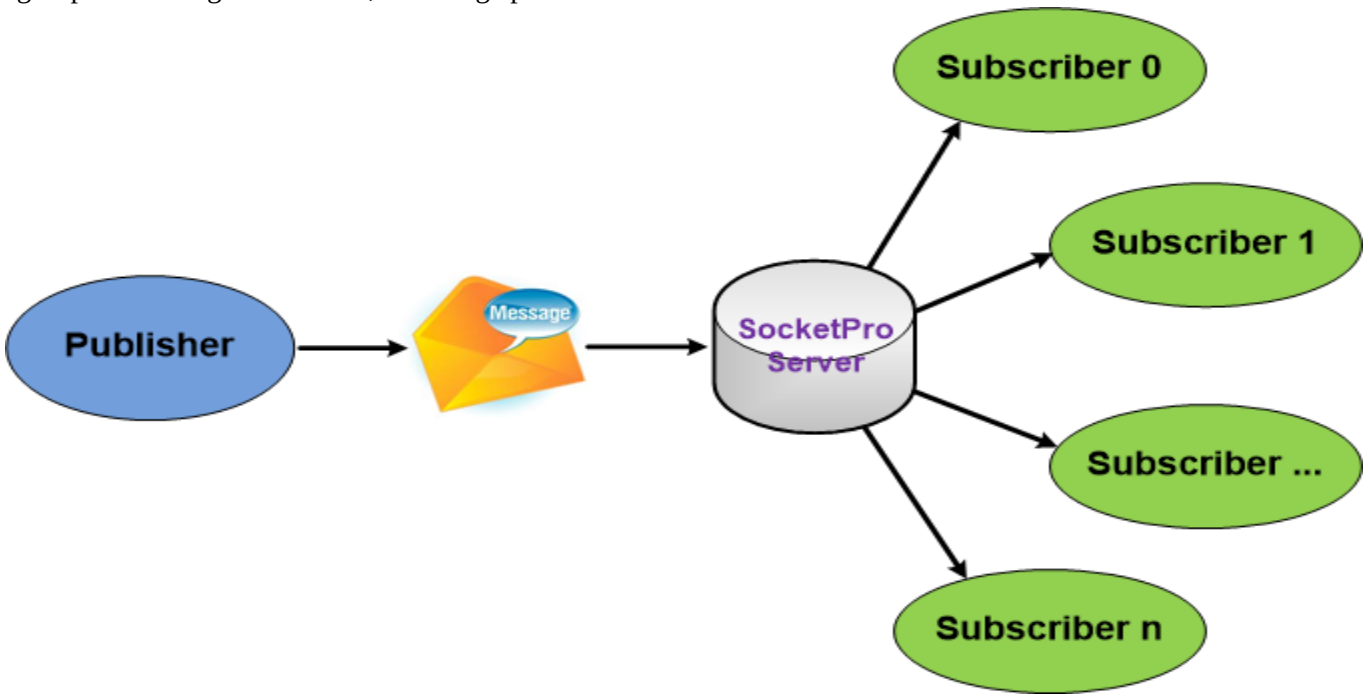

Figure 3: One-to-many real-time notification within SocketPro server

One-to-one: As shown in the Figure 4, SocketPro is also able to send a message from a client to another client with an input receiver user id (case-insensitive).

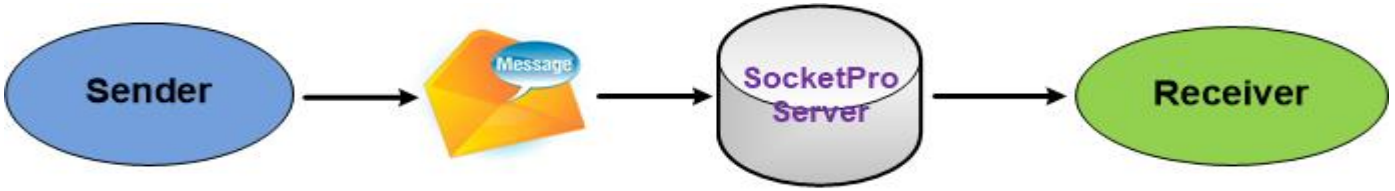

Figure 4: One-to-one real-time notification within SocketPro server

Note that a sender is always able to send a message to a receiver without requirement of any type of subscription, contrary to the previous one-to-many notification model.

Register a few chat groups at server side: Under many cases, we like to use the one-to-many notification model. To do so, we must create a few chat groups or topics at the server side as shown below in Figure 5 before starting SocketPro server.

```
7   protected override bool OnSettingServer()
8   {
9       //amIntegrated and amMixed not supported yet
10      Config.AuthenticationMethod = tagAuthenticationMethod.amOwn;
11
12      PushManager.AddAChatGroup(1, "R&D Department");
13      PushManager.AddAChatGroup(2, "Sales Department");
14      PushManager.AddAChatGroup(3, "Management Department");
15
16      return true; //true -- ok; false -- no listening server
17  }
```

Figure 5: Register a few chat groups at server side

We usually create these chat groups or topics within the virtual function *OnSettingServer*. Lines 12 through 14 register three chat groups. Now, a message sender is able to tell a SocketPro server how to distribute a message from a given array of chat group identification numbers.

If you just use one-to-one notification model only, there is no need to register these chat groups at all.

Subscribe for one or more chat groups of messages: To use the one-to-many chat model, a subscriber or listener for messages must subscribe to the expected groups of messages. You can do so from either the client or server side. This tutorial sample demonstrates when

subscribing for two chat groups of message from the server side as shown in the below Figure 6. We let you find how to subscribe chat groups of messages from the client side after you go through this tutorial.

```
24      protected override void OnSwitchFrom(uint oldServiceId)
25      {
26          uint[] chat_groups = { 1, 3 };
27          Push.Subscribe(chat_groups);
28      }
```

Figure 6: Subscribe two chat groups of message at line 27 at server side

Since this tutorial server is able to support both one-to-many and one-to-one notification models, we need demos to track notification events at both client and server sides.

Track notification events at server side: As shown in the below Figure 7, SocketPro uses virtual functions to track notification events. Therefore, you need to override these virtual functions.

```
30      protected override void OnSubscribe(uint[] groups)
31      {
32          Console.WriteLine(UID + " subscribes for groups " + ToString(groups));
33      }
34
35      protected override void OnUnsubscribe(uint[] groups)
36      {
37          Console.WriteLine(UID + " unsubscribes for groups " + ToString(groups));
38      }
39
40      protected override void OnPublish(object message, uint[] groups)
41      {
42          Console.WriteLine(UID + " publishes a message (" + message + ") to groups " + ToString(groups));
43      }
44
45      protected override void OnSendUserMessage(string receiver, object message)
46      {
47          Console.WriteLine(UID + " sends a message (" + message + ") to " + receiver);
48      }
```

Figure 7: Track common notification events at server side by overriding a set of virtual functions OnXXX

Lines 45 through 48 are used for tracking one-to-one message model messages. All other lines of codes are for tracking message events related to the one-to-many notification model.

Track notification events at client side: At client side, SocketPro uses events approach instead. See Figure 8 below for details.

```
48      CClientSocket ClientSocket = hw.AttachedClientSocket;
49      ClientSocket.Push.OnSubscribe += (cs, messageSender, groups) => {
50          Console.WriteLine("Subscribe for " + ToString(groups));
51          Console.WriteLine(ToString(messageSender));
52          Console.WriteLine();
53      };
54
55      ClientSocket.Push.OnUnsubscribe += (cs, messageSender, groups) => {
56          Console.WriteLine("Unsubscribe from " + ToString(groups));
57          Console.WriteLine(ToString(messageSender));
58          Console.WriteLine();
59      };
60
61      ClientSocket.Push.OnPublish += (cs, messageSender, groups, msg) => {
62          Console.WriteLine("Publish to " + ToString(groups));
63          Console.WriteLine(ToString(messageSender));
64          Console.WriteLine("message = " + msg);
65          Console.WriteLine();
66      };
67
68      ClientSocket.Push.OnSendUserMessage += (cs, messageSender, msg) => {
69          Console.WriteLine("SendUserMessage");
70          Console.WriteLine(ToString(messageSender));
71          Console.WriteLine("message = " + msg);
72          Console.WriteLine();
73      };
```

Figure 8: Track common notification messages at client side

Similarly, to the server side, lines 68 through 73 in the above Figure 8 are used for tracking one-to-one message model messages. All other lines of codes are for tracking message events related with the one-to-many notification model.

4. Send messages

At last, we need some ways to send a message to one or more subscribers (or listeners) as long as there is an interesting thing happening. SocketPro supports sending a message at either client or server side.

Server side: You can call the method *Publish* after obtaining the property *Push* as shown in the Figure 9.

```
56          //notify a message to groups [2, 3] at server side
57          Push.Publish("Say hello from " + firstName + " " + lastName, 2, 3);
```

Figure 9: Send a message onto two chat groups of subscribers at server side

Similarly, you can send a message to a receiver identified by its user id with the method *SendUserMessage*.

Client side: You are still able to use the class *CClientSocket* property *Push* to send your interesting messages as shown in the below Figure 10.

```
79          //process multiple requests in batch asynchronously
80          ok = hw.StartBatching();
81          ok = hw.SendRequest(hwConst.idSayHelloHelloWorld, "Jack", "Smith", (ar) =>
82          {
83              string ret;
84              ar.Load(out ret);
85              Console.WriteLine(ret);
86          });
87
88          uint[] chat_ids = { 1, 2 };
89          ok = ClientSocket.Push.Publish("We are going to call the method Sleep", chat_ids);
90          CAsyncServiceHandler.DAsyncResultHandler arh = null;
91          ok = hw.SendRequest(hwConst.idSleepHelloWorld, (int)5000, arh);
92
93          Console.WriteLine("Input a receiver for receiving my message ......");
94          Console.WriteLine();
95          ok = ClientSocket.Push.SendUserMessage("A message from " + cc.UserId, Console.ReadLine());
96
97          ok = hw.CommitBatching(false); //true -- ask server return multiple results in one shot
98          ok = hw.WaitAll();
```

Figure 10: Send messages at client side

Figure 10 demonstrates publishing a message from line 89 to two chat groups of subscribers, and then sending one message to one receiver as identified by an input user id at line 95.