Base Language - Feature #4366

support TLS 1.2 and other updated secure socket protocols

10/22/2019 03:25 PM - Greg Shah

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Roger Borrello	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Base Language - Feature #2181: implement SSL support for sockets	**Closed**
Related to Base Language - Feature #5662: add -WSDLAuth and -SSLAuth (and rel...	**New**

History

#1 - 10/22/2019 03:29 PM - Greg Shah

LowLevelSocketImpl.createSslSocket() uses SSLContext.getInstance("SSL") to create the SSL environment. This supports a very basic level of the protocol, but we need to enable all recent protocol levels including TLS 1.2. JDK8 already supports this, but we must allow different protocol names to be used to create the context.

If there is a way that the 4GL can specify this, we should support it. We also should allow more recent protocol names if the 4GL has limits which Java can transcend.

#2 - 12/04/2020 10:16 AM - Greg Shah

My understanding is that this support is needed for the "socket" (CREATE SOCKET) and "server socket" (CREATE SERVER-SOCKET) handle-based objects. Details of how to write code for sockets are documented in Chapter 20 ("Sockets") of the *OpenEdge External Programming Interface Reference*. The most obvious way to support this is to add new syntax to the "connect string". This is the parameter passed to <SERVER_SOCKET_HANDLE>:ENABLE-CONNECTIONS("connect_parms") or <SOCKET_HANDLE>:CONNECT("connect_parms") handle-based method calls. Such an approach would require no conversion changes and would only need changes in the runtime to parse out our "enhanced" connection option(s) for the new SSL/TLS protocol strings.

I'm confirming with the customer (who requested this) right now.

LE: The customer confirmed this is correct.

#3 - 12/08/2020 09:39 AM - Greg Shah

- Assignee set to Roger Borrello

#4 - 04/19/2021 10:34 AM - Roger Borrello

Greg Shah wrote:

> My understanding is that this support is needed for the "socket" (CREATE SOCKET) and "server socket" (CREATE SERVER-SOCKET)

handle-based objects. Details of how to write code for sockets are documented in Chapter 20 ("Sockets") of the *OpenEdge External Programming Interface Reference*. The most obvious way to support this is to add new syntax to the "connect string". This is the parameter passed to <SERVER_SOCKET_HANDLE>:ENABLE-CONNECTIONS("connect_parms") or <SOCKET_HANDLE>:CONNECT("connect_parms") handle-based method calls. Such an approach would require no conversion changes and would only need changes in the runtime to parse out our "enhanced" connection option(s) for the new SSL/TLS protocol strings.

4GL specifies this parameter for <SERVER_SOCKET_HANDLE>:ENABLE-CONNECTIONS("connect_parms"): -ssl = If specified, the connection is SSL-based. and <SOCKET_HANDLE>:CONNECT("connect_parms"): -ssl = If specified, the connection to the server socket uses Secure Sockets Layer (SSL) tunneling.

The current -ssl option retrieves additional parameters:
-keyalias (string="default_server"
-keyaliaspasswd (string)
-nosessioncache (boolean)
-sessiontimeout (integer)

Given the below table, we could include the additional names to the connect_parms, but we'll need to formulate any additional syntax for parameters based upon the indicated algorithm:

Algorithm Name	Description
SSL	Supports some version of SSL; may support other versions
SSLv2	Supports SSL version 2 or later; may support other versions
SSLv3	Supports SSL version 3; may support other versions
TLS	Supports some version of TLS; may support other versions
TLSv1	Supports RFC 2246: TLS version 1.0 ; may support other versions
TLSv1.1	Supports RFC 4346: TLS version 1.1 ; may support other versions
TLSv1.2	Supports RFC 5246: TLS version 1.2 ; may support other versions

#5 - 08/31/2021 04:59 PM - Roger Borrello

Does 4GL already have syntax to enable this? Or are we making up our own?

In looking for the best documentation, it's pretty pathetic:

https://docs.progress.com/bundle/openedge-programming-interfaces/page/Sockets.html
https://documentation.progress.com/output/ua/OpenEdge_latest/pdsoe/PLUGINS_ROOT/com.openedge.pdt.langref.help/rfi1424919881995.html
https://docs.progress.com/bundle/openedge-programming-interfaces/page/Enabling-TLS-server-connections.html
https://docs.progress.com/bundle/openedge-programming-interfaces/page/Implementing-an-ABL-socket-server.html#Implementing-an-ABL-socket-server

#6 - 09/01/2021 06:38 AM - Greg Shah

It is our own syntax. This is a new feature/enhancement. Just make the syntax anything that makes sense and is easy to support/maintain.

#7 - 09/01/2021 06:39 AM - Greg Shah

And which doesn't conflict with existing syntax from the 4GL.

#8 - 09/01/2021 01:02 PM - Roger Borrello

The testcases I found in uast/sockets and uast/sockets/ssl are very helpful. However, I may need a little help configuring the ssl version.

In LowLevelSocketListener:enableSSLConnections, I am allowing the keyalias and keyaliaspasswd to default to default_server and password, respectively. My configuration seems to lead to NPE when:

```
      // open key store, check the existence of the requested alias
      KeyStore ks = KeyStore.getInstance("JKS");                  <-- I believe this comes back null
      ks.load(new FileInputStream(keyStoreName), keyStorePass);
      if (!ks.isKeyEntry(keyalias))
      {
         // requested alias was not found in keystore
         return -1;
      }
```

Is there something I should have in the directory, or on the file system?

#9 - 09/01/2021 02:42 PM - Greg Shah

KeyStore ks = KeyStore.getInstance("JKS"); <-- I believe this comes back null

That doesn't sound right. The "JKS" type is well supported and even if it wasn't supported you should see a KeyStoreException instead of a null return value.

There is nothing special that is needed to have access to the "JKS" type. The provider is built into the JRE.

#10 - 09/01/2021 02:45 PM - Roger Borrello

Actually, the keyStoreName cannot be found. I tried to copy ./deploy/server/standard-private-key.store to default_server.store and placed it in various directories, as well as part of the jar in one of the search directories (./sockets) and it isn't ever found.

```
      String keyStoreName = ThinClient.getLocalFileName(keyalias + ".store");  <-- Comes back null for "defaul
t_server.store"

      // file-level password for keystore. Same for all stores. The password is passed in as
      // a bootstrap parameter value with "ssl-socket:keystore:password" key
      char keyStorePass[] = keyStorePassword.toCharArray();

      // keyaliaspasswd is already decrypted on p2j-server side
      char ctPass[] = keyaliaspasswd.toCharArray();

      // open key store, check the existence of the requested alias
      KeyStore ks = KeyStore.getInstance("JKS");
      ks.load(new FileInputStream(keyStoreName), keyStorePass);  <-- NPE here because keyStoreName is null
```

#11 - 09/01/2021 02:47 PM - Constantin Asofiei

I suspect the correct location is in the deploy/client folder or wherever the FWD client working dir is.

#12 - 09/01/2021 03:08 PM - Roger Borrello

Constantin Asofiei wrote:

> I suspect the correct location is in the deploy/client folder or wherever the FWD client working dir is.

That worked. However, I created a tampered store: java.io.IOException: Keystore was tampered with, or password was incorrect

I have to pass -keyalias to an actual store with -keyaliaspasswd to an encrypted password, or generate default_server.store with the password as password.

#13 - 09/01/2021 03:15 PM - Greg Shah

Please see #2181 for details on the existing 4GL-compatible SSL socket support. Search on "keystore" and "default_server". Try not to get too hung up on this. We just want to use the keystore support as already implemented, but allow newer TLS levels for the socket itself.

#14 - 09/01/2021 03:15 PM - Greg Shah

- Related to Feature #2181: implement SSL support for sockets added

#15 - 09/01/2021 06:40 PM - Roger Borrello

Greg Shah wrote:

> Please see #2181 for details on the existing 4GL-compatible SSL socket support. Search on "keystore" and "default_server". Try not to get too hung up on this. We just want to use the keystore support as already implemented, but allow newer TLS levels for the socket itself.

It's not a question of getting hung up. I would expect that for any algorithm I add, there will be a certificate built appropriately to allow me to test a connection. So getting at least the lowest level correct seems needed. While that task was very helpful, I tried to put together the steps, but I still end up unsuccessful starting the server task:

```
java.io.IOException: keystore password was incorrect
        at sun.security.pkcs12.PKCS12KeyStore.engineLoad(PKCS12KeyStore.java:2068)
        at sun.security.provider.KeyStoreDelegator.engineLoad(KeyStoreDelegator.java:238)
        at sun.security.provider.JavaKeyStore$DualFormatJKS.engineLoad(JavaKeyStore.java:71)
        at java.security.KeyStore.load(KeyStore.java:1445)
```

I used these steps:

```
 $ keytool -genkey -keyalg RSA -alias default_server -keystore default_server.store -storepass <password> -val
idity 48 -keysize 1024
 $ keytool -exportcert -keystore default_server.store -alias default_server > default_server.cer
 $ keytool -import -alias default_server -file default_server.cer -keystore trusted-cert.store
```

I just allowed all the Unknown entries, and tried setting the password to password.

I was presented with a warning on the -exportcert:

```
Warning:
The certificate uses a 1024-bit RSA key which is considered a security risk. This key size will be disabled in
 a future update.
The JKS keystore uses a proprietary format. It is recommended to migrate to PKCS12 which is an industry standa
rd format using "keytool -importkeystore -srckeystore default_server.store -destkeystore default_server.store
 -deststoretype pkcs12".
```

I also tried to update the store using the command given, but still get the bad password error.

#16 - 09/02/2021 07:54 AM - Ovidiu Maxiniuc

Roger Borrello wrote:

> It's not a question of getting hung up. I would expect that for any algorithm I add, there will be a certificate built appropriately to allow me to test a connection. So getting at least the lowest level correct seems needed. While that task was very helpful, I tried to put together the steps, but I still end up unsuccessful starting the server task:
>
> ```
> java.io.IOException: keystore password was incorrect
> [...]
> ```

The key is incorrect because it was not passed in at startup, see LowLevelSocketListenerImpl.java:262:

```
    // file-level password for keystore. Same for all stores. The password is passed in as
    // a bootstrap parameter value with "ssl-socket:keystore:password" key
    char[] keyStorePass = keyStorePassword.toCharArray();
```

Most likely at this point keyStorePassword was an empty String.

#17 - 09/02/2021 11:46 AM - Roger Borrello

Ovidiu Maxiniuc wrote:

> Most likely at this point keyStorePassword was an empty String.

Thanks, Ovidiu. That is the case. I am trying to follow the Wiki for certificates just to utilize what I already have configured in the directory, but if there's a place where I am supposed to specify "standard" instead of "default_server" for the store, so as to get the little test application to use them, it would be very helpful to get an example. Specifying **-keyaliaspasswd password** to the ENABLE-CONNECTIONS doesn't make any difference.

#18 - 09/02/2021 02:25 PM - Roger Borrello

When -ssl is passed in the ENABLE-CONNECTIONS string LowLevelSocketListerner.enableSSLConnections performs:

```
    // create the SSL context
    SSLContext sslCtx = SSLContext.getInstance("TLS");
```

My plan was to allow any of these to be passed into the ENABLE-CONNECTIONS or CONNECT string, and generate the appropriate SSLContext:

However, now that I see **TLS** is the context passed to the getInstance method, perhaps we should just have -tls as an additional option, which would generate the TLSv1.2 context? The Java 8 documentation for SSLContext indicates that (https://docs.oracle.com/javase/8/docs/api/javax/net/ssl/SSLContext.html):

> Every implementation of the Java platform is required to support the following standard SSLContext protocol:
>
> - TLSv1

#19 - 09/02/2021 02:31 PM - Greg Shah

> perhaps we should just have -tls as an additional option, which would generate the TLSv1.2 context?

No, I think this "hides" things and is not as clear.

I like your original idea of providing a range of string parameters. If the chosen secure socket context is not valid, then the enable-connections or connect() should fail.

#20 - 09/02/2021 03:23 PM - Roger Borrello

Greg Shah wrote:

> I like your original idea of providing a range of string parameters. If the chosen secure socket context is not valid, then the enable-connections or connect() should fail.

I agree. However, if the -ssl option results in TLS SSLContext, perhaps I just provide -tlsv1, -tlsv1.1 and -tlsv1.2 and if there's any conflicts, opt for the highest level of support.

#21 - 09/02/2021 05:33 PM - Roger Borrello

I have most all of this task coded up, but I am unsure of one aspect. I have added a **String protocol** parameter that is determined in the SocketImpl.java and SocketListernerImpl.java. Basically:

```
      boolean ssl    = parms.containsKey("-ssl");
      boolean tlsv11 = parms.containsKey("-tlsv11");
      boolean tlsv12 = parms.containsKey("-tlsv12");
      String protocol = "TLS";
      if (tlsv11)
      {
          protocol = "TLSv1.1";
      }
      else if (tlsv12)
      {
          protocol = "TLSv1.2";
      }
```

This will utilize the existing "TLS" value, unless the -tlsv11 or -tlsv12 parameters are passed. It was straightforward to include this as new parameter to all the other methods, except for where LowLevelSocketImpl.initialize overrides the createSocket and calls createSslSocket. I haven't determined where the initialize method gets called from, so for now I just passed "TLS" in:

```
            return createSslSocket(host,
```

```
                                    port,
                                    inetAddress,
                                    localPort,
                                    params.getConnectionTimeout(),
                                    noHostVerify,
                                    false,
                                    "TLS");
```

I can check everything in for review, but I'd like to know how this instance is invoked, because it might be before we determine if any parameters are passed from the 4GL procedures.

#22 - 09/03/2021 07:27 AM - Greg Shah

> I have most all of this task coded up, but I am unsure of one aspect. I have added a String protocol parameter that is determined in the SocketImpl.java and SocketListernerImpl.java. Basically:

This seems fine.

> I haven't determined where the initialize method gets called from,

In Eclipse, you can navigate to the LowLevelSocketImpl.initialize() in the code. Double click on the method name initialize in the method signature itself and then press CTRL-ALT-H. This will show you the "call hierarchy" which is the locations in the code where there are hard coded calls to that method. You can see that it is only called from ThinClient.initializePost().

> so for now I just passed "TLS" in:

ThinClient.initializePost() is only called once at startup of the session so it is not suitable for passing that value since it would be hard coded for all connections. You will need to come up with a mechanism that allows us to pass parameters in during every connection. The HttpConnectionParams is the way to do this. You can see this technique in WebServiceImpl.invoke(), search on P2J_NOHOSTVERIFY.

#23 - 09/03/2021 08:55 AM - Roger Borrello

Thanks for the tips... I'll take a look.

#24 - 09/03/2021 09:06 AM - Roger Borrello

Greg Shah wrote:

> ThinClient.initializePost() is only called once at startup of the session so it is not suitable for passing that value since it would be hard coded for all connections. You will need to come up with a mechanism that allows us to pass parameters in during every connection. The HttpConnectionParams is the way to do this. You can see this technique in WebServiceImpl.invoke(), search on P2J_NOHOSTVERIFY.

So we are going to introduce TLSv1.1 and TLSv1.2 to the web services connection? Currently it is using "TLS". How/where would the value be specified?

#25 - 09/03/2021 10:19 AM - Roger Borrello

Ovidiu, Igor, is there any need to allow the custom SSLProtocolSocketFactory defined in the LowLevelSocketImpl.initialize to specify a higher protocol than TLS? Currently the overriden createSocket was not specifying protocol, and the createSslSocket method was simply passing "TLS" to the SSLContext.getInstance("TLS") method. With other changes for the 4GL enhancements, we are determining and passing other protocols. In order to do so with the initialize, there are additional "hoops" to jump through. But is it a necessary addition, in your opinion? Or can I just leave it as "TLS".

#26 - 09/03/2021 10:28 AM - Igor Skornyakov

Roger Borrello wrote:

> Ovidiu, Igor, is there any need to allow the custom SSLProtocolSocketFactory defined in the LowLevelSocketImpl.initialize to specify a higher protocol than TLS? Currently the overriden createSocket was not specifying protocol, and the createSslSocket method was simply passing "TLS" to the SSLContext.getInstance("TLS") method. With other changes for the 4GL enhancements, we are determining and passing other protocols. In order to do so with the initialize, there are additional "hoops" to jump through. But is it a necessary addition, in your opinion? Or can I just leave it as "TLS".

Roger,
As far as I know the TLS versions prior to 1.2 are considered obsolete now. Since our SSL support is based on the JDK/BC SSLEngine and the exact TLS version used in a session is a result of the SSL handshake negotiation I think we can rely on them and do not need to care about the version unless we have some special reasons to insist on using the old versions.
In my tests, TLS 1.2 was always selected.

#27 - 09/03/2021 12:46 PM - Roger Borrello

Igor Skornyakov wrote:

> Roger,
> As far as I know the TLS versions prior to 1.2 are considered obsolete now. Since our SSL support is based on the JDK/BC SSLEngine and the exact TLS version used in a session is a result of the SSL handshake negotiation I think we can rely on them and do not need to care about the version unless we have some special reasons to insist on using the old versions.
> In my tests, TLS 1.2 was always selected.

So as long as we use SSLContext.getInstance("TLS") we get the highest TLS level (not the lowest)?

#28 - 09/03/2021 12:51 PM - Igor Skornyakov

Roger Borrello wrote:

> Igor Skornyakov wrote:
>
> > Roger,
> > As far as I know the TLS versions prior to 1.2 are considered obsolete now. Since our SSL support is based on the JDK/BC SSLEngine and the exact TLS version used in a session is a result of the SSL handshake negotiation I think we can rely on them and do not need to care about the version unless we have some special reasons to insist on using the old versions.
> > In my tests, TLS 1.2 was always selected.
>
> So as long as we use SSLContext.getInstance("TLS") we get the highest TLS level (not the lowest)?

It depends on the Java security configuration on both sides. See for example <jre home>/lib/security/java.security file. With default settings, it will be TLS 1.2.

#29 - 09/03/2021 01:54 PM - Greg Shah

Perhaps we should force it to the 1.2 level. Since a v1 or v1.1 peer can negotiate the connection down to the older protocol, this is safe, right? My concern is that we don't want the default version controlled at runtime by the user's environment.

The only downside is that future Java versions that support later TLS levels (past 1.2) will need changes in FWD to support them.

#30 - 09/07/2021 10:21 AM - Roger Borrello

Greg Shah wrote:

> Perhaps we should force it to the 1.2 level. Since a v1 or v1.1 peer can negotiate the connection down to the older protocol, this is safe, right? My concern is that we don't want the default version controlled at runtime by the user's environment.
>
> The only downside is that future Java versions that support later TLS levels (past 1.2) will need changes in FWD to support them.

Could we just document the possibilities, and stick with the latest?

#31 - 09/07/2021 10:56 AM - Greg Shah

First, let's note that the various versions of SSL are not the same as the various TLS versions. There are slight incompatibilities between them. The 4GL "-ssl" connect option uses SSL, as far as I understand. It is not clear which version of SSL they support. Hopefully it is at least SSL 3.0.

While TLS 1.2 may gracefully downgrade to SSL 3.0 (but not older SSL versions), TLS 1.3 is not planned to downgrade because downgrading is dangerous. The short story there is that a client that can force a downgrade in protocol can open up a security hole when someone exploit security vulnerabilities in that older protocol.

The current code uses SSLContext.getInstance("SSL"), which is NOT TLS (see here). Regardless of the possibility of TLS sometimes downgrading and possibly being compatible, I don't think we can rely upon that, especially in the forthcoming TLS 1.3. Thus we can't simply implement TLS when the connect option is -ssl. For compatibility, we need to continue mapping -ssl to SSLContext.getInstance("SSL").

But we can add -tls and map that to SSLContext.getInstance("TLS"), which for now will provide TLS 1.2 support. My concern here is what happens later, when TLS 1.3 becomes the default. Then existing implementations may fail if the clients require TLS 1.2.

Please implement the full set of options from #4366-18 and I think we are covered. Customers can use -tls for systems that have flexibility and should "auto-upgrade to the latest level". But they will have control over the specific levels and can ensure compatibility when needed.

#32 - 09/07/2021 02:23 PM - Roger Borrello

In examining the difference between the src/com/goldencode/p2j/util/LowLevelSocketImpl.java and src/com/goldencode/p2j/util/LowLevelSocketListenerImpl.java a little closer, I see that LowLevelSocketImpl.createSslSocket has:

```
         SSLContext sslCtx = SSLContext.getInstance("SSL"); // "TLSv1"
```

while LowLevelSocketListenerImpl.enableSSLConnections has:

```
      SSLContext sslCtx = SSLContext.getInstance("TLS");
```

Does that affect anything, that the listener is using "SSL", while the creator is using "TLS"?

#33 - 09/07/2021 02:34 PM - Greg Shah

I don't know for sure, but I suspect that LowLevelSocketListenerImpl.enableSSLConnections() should be using "SSL".

#34 - 09/07/2021 03:08 PM - Roger Borrello

Greg Shah wrote:

> I don't know for sure, but I suspect that LowLevelSocketListenerImpl.enableSSLConnections() should be using "SSL".

With this implementation, we should be covered:

```
boolean ssl    = parms.containsKey("-ssl");
boolean sslv2  = parms.containsKey("-sslv2");
boolean sslv3  = parms.containsKey("-sslv3");
boolean tls    = parms.containsKey("-tls");
boolean tlsv11 = parms.containsKey("-tlsv11");
boolean tlsv12 = parms.containsKey("-tlsv12");
String protocol = "SSL";
if (sslv2)
{
    protocol = "SSLv2";
}
else if (sslv3)
{
    protocol = "SSLv3";
}
else if (tlsv11)
{
    protocol = "TLSv1.1";
}
else if (tlsv12)
{
    protocol = "TLSv1.2";
}
else if (tls)
{
    protocol = "TLS";
}
```

This would cover those connections, but we still need a way to determine what to pass in the #4366-21 when LowLevelSocketImpl.initialize creates the socket.

but we still need a way to determine what to pass in the #4366-21 when LowLevelSocketImpl.initialize creates the socket.

Secure sockets can be created/initialized in 5 places from 4GL code:

- 4 possible CONNECT() cases
  - SOCKET:CONNECT()
  - WEB-SERVICE:CONNECT()
  - appserver SERVER-OBJECT:CONNECT()
  - PASOE SERVER-OBJECT:CONNECT())
- SERVER-SOCKET:ENABLE-CONNECTIONS()

SOCKET:CONNECT

On the server, a call to <socket_handle>:CONNECT() is implemented in SocketImpl.connect(). This is the place where we parse out the connect options. For the -ssl case we call over to the client side using LowLevelSocketImpl.connectSSL(). At this time we don't pass a specific type of SSL session. You will have to add that. The definition of that interface is in LowLevelSocket.connectSSL(). Inside LowLevelSocketImpl.connectSSL(), the code directly calls LowLevelSocketImpl.createSslSocket() and you will have to add the parameter there as well. Inside LowLevelSocketImpl.createSslSocket() is the actual call to SSLContext.getInstance("SSL") which will have its type passed in instead of hard coded.

WEB-SERVICE:CONNECT

This is a second case that uses the same path as SOCKET:CONNECT(). On the server, a call to <web_service_handle>:CONNECT() will pass -WSDLAuth (as ssl) or -SSLAuth (as ssl) to force SSL. I don't see where the FWD code actually honors those speciifc parameters (see WebServiceHelper.knownConnectOptions). Anyway, the converted 4GL code will call into ServerImpl.connect() which will use ServerImpl.parseOptions() to handle the options processing. Then it will call ServerImpl.connectToWebService() which will call WebServiceHelper.connect() which calls the client-side WebServiceImpl.connect(). I think that that code builds up all the web services gorp that will eventually use the SSLProtocolSocketFactory that is created by LowLevelSocketImpl.initialize() code. That factory instance overrides the createSocket() in the client side implementation and inside that it calls LowLevelSocketImpl.createSslSocket(). The options processing is pretty opaque. It seems to use some "magic glue" in WebServiceImpl.WebServiceData to handle the options includng the nohostverify. To handle different SSL types, we would have to live within that structure. Today that code uses "SSL" because it is hard coded in LowLevelSocketImpl.createSslSocket(). It would be safe to code it that way and it may be needed until the moment we actually honor the -WSDLAuth or -SSLAuth (and related authentication) options.

Constantin: I did not realize (or don't remember) that our support is incomplete in this area. Am I missing something?

SERVER-OBJECT:CONNECT

The APPSERVER <server_object>:CONNECT() use of -ssl is not directly honored since this is "appserver" usage and that means we are calling converted 4GL code in a FWD server. In that case we always use the secure transport built into FWD so there is no separate SSL setup. There is nothing to do for this case.

The PASOE <server_object>:CONNECT() doesn't support -ssl but does have some protocol options. But again, this is really the same thing as the appserver call above and it just uses the secure transport in FWD. There is nothing to do for this case.

SERVER-SOCKET:ENABLE-CONNECTIONS

On the server side, the 4GL code for <server_socket_handle>:ENABLE-CONNECTIONS() is implemented in SocketListenerImpl.enableConnections(). It calls parseOptions() to check on the -ssl parameter. For the SSL case, it then calls the client side LowLevelSocketListenerImpl.enableSSLConnections() which currently is hard coded to call SSLContext.getInstance("TLS"). At this time we don't pass a specific type of SSL session. You will have to add that. The definition of that interface is in LowLevelSocketListener.enableSSLConnections().

Although I have open questions above for Constantin, I hope it is clear that for that web services case, you should hard code "SSL". The 2 other cases should be augmented properly as noted above.

#36 - 09/08/2021 04:25 AM - Constantin Asofiei

Greg Shah wrote:

> WEB-SERVICE:CONNECT
>
> On the server, a call to <web_service_handle>:CONNECT() will pass -WSDLAuth (as ssl) or -SSLAuth (as ssl) to force SSL. I don't see where the FWD code actually honors those speciifc parameters (see WebServiceHelper.knownConnectOptions).

The web server support (for server handle) was added with 10.2B specs IIRC. These -WSDLAuth and -SSLAuth seem to be added in a later OE version, they do not appear in the 10.2B documentation.

> Anyway, the converted 4GL code will call into ServerImpl.connect() which will use ServerImpl.parseOptions() to handle the options processing. Then it will call ServerImpl.connectToWebService() which will call WebServiceHelper.connect() which calls the client-side WebServiceImpl.connect().

This is correct, but this just saves all the configuration for this web service connection - it does not establish a persistent HTTP connection at this time.

> I think that that code builds up all the web services gorp that will eventually use the SSLProtocolSocketFactory that is created by LowLevelSocketImpl.initialize() code.

No, there is no persistent connection to the remote web server. The connection is established on each call, see WebServiceImpl.invoke, which executes the SOAP call using a axis2 ServiceClient.

> That factory instance overrides the createSocket() in the client side implementation and inside that it calls LowLevelSocketImpl.createSslSocket(). The options processing is pretty opaque. It seems to use some "magic glue" in WebServiceImpl.WebServiceData to handle the options includng the nohostverify. To handle different SSL types, we would have to live within that structure. Today that code uses "SSL" because it is hard coded in LowLevelSocketImpl.createSslSocket(). It would be safe to code it that way and it may be needed until the moment we actually honor the -WSDLAuth or -SSLAuth (and related authentication) options.

This is not correct, there is no LowLevelSocketImpl usage for the WebServiceImpl at this time.

I don't know why I set sc.getOptions().setProperty(LowLevelSocketImpl.P2J_NOHOSTVERIFY, wsd.isNoHostVerify()); as this has no effect for axis2's ServiceClient.

I think the correct solution is to build our own org.apache.commons.httpclient.HttpClient instance and set that to ServiceClient via ServiceClient.getOptions().setProperty(.org.apache.axis2.transport.http.HTTPConstants.CACHED_HTTP_CLIENT, customHttpClientInstace). This will override the default HttpClient instance.

#37 - 09/08/2021 07:33 AM - Greg Shah

> This is not correct, there is no LowLevelSocketImpl usage for the WebServiceImpl at this time.

Does anyone know what code uses the registered SSLProtocolSocketFactory?

> I think the correct solution is to build our own org.apache.commons.httpclient.HttpClient instance and set that to ServiceClient via ServiceClient.getOptions().setProperty(.org.apache.axis2.transport.http.HTTPConstants.CACHED_HTTP_CLIENT, customHttpClientInstace). This will override the default HttpClient instance.

Can we postpone adding this until we add support for the -WSDLAuth and -SSLAuth (and related) connect options? Or is there some other benefit to implementing this solution?

#38 - 09/08/2021 08:18 AM - Constantin Asofiei

Greg Shah wrote:

> > This is not correct, there is no LowLevelSocketImpl usage for the WebServiceImpl at this time.
>
> Does anyone know what code uses the registered SSLProtocolSocketFactory?

Argh, I see it now, I thought you meant explicit usage for LowLevelSocketImpl in WebServiceImpl (as in, the web service connection relying on the legacy OE client socket implemnetation in LowLevelSocketImpl).

That code in LowLevelSocketImpl.initialize I think affects any http connection established via a HttpClient (including Axis2). Now it makes sense why sc.getOptions().setProperty(LowLevelSocketImpl.P2J_NOHOSTVERIFY, wsd.isNoHostVerify()); was set. I don't really like it because it doesn't allow a more granular configuration, like overriding at runtime the certificate stores and such.

> > I think the correct solution is to build our own org.apache.commons.httpclient.HttpClient instance and set that to ServiceClient via ServiceClient.getOptions().setProperty(.org.apache.axis2.transport.http.HTTPConstants.CACHED_HTTP_CLIENT, customHttpClientInstace). This will override the default HttpClient instance.
>
> Can we postpone adding this until we add support for the -WSDLAuth and -SSLAuth (and related) connect options? Or is there some other benefit to implementing this solution?

Yes, we can postpone any refactoring of WebServiceImpl until these are added. SSL is working at this time.

#39 - 09/08/2021 12:26 PM - Greg Shah

- Related to Feature #5662: add -WSDLAuth and -SSLAuth (and related) connect options for the web-services handle added

#40 - 09/08/2021 03:27 PM - Roger Borrello

- % Done changed from 0 to 100

- Status changed from New to Feedback

Please review:

```
modified src/com/goldencode/p2j/util/LowLevelSocketListener.java
modified src/com/goldencode/p2j/util/LowLevelSocket.java
modified src/com/goldencode/p2j/util/LowLevelSocketListenerImpl.java
modified src/com/goldencode/p2j/util/SocketImpl.java
modified src/com/goldencode/p2j/util/LowLevelSocketImpl.java
modified src/com/goldencode/p2j/util/SocketListenerImpl.java
Committed revision 12901.
```

#41 - 09/08/2021 04:21 PM - Greg Shah

- Status changed from Feedback to Test

Code Review Task Branch 3821c Revision 12901

The changes look good.

#42 - 09/08/2021 06:20 PM - Roger Borrello

- Assignee changed from Roger Borrello to Eric Faulhaber

- vendor_id deleted (GCD)

I am back to testing with my testcases...the ssl-server.p has:

```
define variable ka   as char init "shared".
define variable kapw as char init "v~~4l#8ZrkkJwW?V681*wl945GbzmHmxc".
.
.
.
hsSocket:ENABLE-CONNECTIONS("-S 9123 -ssl -nosessioncache -sessiontimeout 60 -keyalias " + ka + " -keyaliaspas
swd " + kapw).
```

The shared alias I got from the SSLCertGenUtil output when I ran against the testcases directory.xml:

```
java -classpath ~/projects/fwd/p2jdev/build/lib/p2j.jar com.goldencode.p2j.security.SSLCertGenUtil -dir direct
ory.xml --reuse-passwords no --reuse-ca no -M yes
.
.
.
Private key for alias [shared] was encrypted using the [v~4l#8ZrkkJwW?V681*wl945GbzmHmxc] password and saved i
```

```
n the /security/certificates/private-keys/shared node.
```

But when I get to the decrypt keyaliaspasswd = SymmetricEncryption.decrypt(keyaliaspasswd);, it throws java.lang.NumberFormatException: For input string: "v~"

#43 - 09/08/2021 06:52 PM - Roger Borrello

- Assignee changed from Eric Faulhaber to Roger Borrello

- vendor_id set to GCD

#44 - 09/09/2021 07:15 AM - Greg Shah

The tilde is an escape char in the 4GL. If your code is this:

define variable kapw as char init "v~~4l#8ZrkkJwW?V681*wl945GbzmHmxc".

Then the ~~ will result in a single tilde in the actual string. Is that what you expect? If the actual key password has 2 tildes, then you need to double them in your text like this:

define variable kapw as char init "v~~~~4l#8ZrkkJwW?V681*wl945GbzmHmxc".

#45 - 09/09/2021 08:17 AM - Roger Borrello

Greg Shah wrote:

> Is that what you expect? If the actual key password has 2 tildes, then you need to double them in your text like this:

Yes... there was a tilde in the passcode, so I had to stuff another in the string. In the debugger, it matched up when evaluated.

#46 - 09/09/2021 08:26 AM - Greg Shah

If you look at the code for SymmetricEncryption.decrypt(), you'll see that in asBytes() it assumes that the input text is encoded in hexadecimal format.

#47 - 09/09/2021 09:46 AM - Greg Shah

Ovidiu: Please edit Socket Handle and Server Socket Handle SSL Connections to document the exact steps for the setup of a legacy 4GL socket connection (inbound or outbound).

I think the details in #2181 are not enough for Roger to have gotten a working connection. So there must be some additions needed. Plus, it is not easy to "consume" right now, so real docs are needed.

Roger: Please ask Ovidiu specific questions to get help with this configuration. Also, after Ovidiu has the docs written, please add to that page for the specifics of the extension you've added here (all the new -tls etc... connect option values and what they mean).

#48 - 09/09/2021 06:29 PM - Roger Borrello

Greg Shah wrote:

> Roger: Please ask Ovidiu specific questions to get help with this configuration. Also, after Ovidiu has the docs written, please add to that page for the specifics of the extension you've added here (all the new -tls etc... connect option values and what they mean).

That task was using older utilities, and we should be documenting our current ones. Ovidiu, I was using SSLCertGenUtil to do the generation of the keys into the directory ("shared" alias), but trying to figure out what to pass as the option to -keyaliaspasswd is where my challenge is.

#49 - 09/10/2021 07:04 AM - Greg Shah

Passing the output from the SSLCertGenUtil cannot work since it is not encoded as hex.

Ovidiu: Please document the specific requirements. We may need to make code changes to make it easier to configure and use SSL.

#50 - 09/10/2021 02:03 PM - Ovidiu Maxiniuc

Roger Borrello wrote:

> Ovidiu, I was using SSLCertGenUtil to do the generation of the keys into the directory ("shared" alias), but trying to figure out what to pass as the option to -keyaliaspasswd is where my challenge is.

Try passing the password through SymmetricEncryption.encrypt(String).

#51 - 09/10/2021 02:17 PM - Ovidiu Maxiniuc

Greg Shah wrote:

> Ovidiu: Please document the specific requirements. We may need to make code changes to make it easier to configure and use SSL.

I added content to Socket Handle and Server Socket Handle SSL Connections. I documented the exact steps to create a simple plain-text and SSL-enabled server and a client to connect to it in 4GL. Then the configuration updates to run the same code, converted, in FWD.

I tried to find the commands for converting the key/certificates from .pem to JKS format but I am blocked here because of some exceptions along the way. The best suited tutorial for this I found here: https://docs.oracle.com/cd/E35976_01/server.740/es_admin/src/tadm_ssl_convert_pem_to_jks.html . This is because I think a customer may want to keep their old resources (even if in another format) with the converted code instead of generating new keys from the scratch.

#52 - 09/10/2021 03:49 PM - Greg Shah

In your text, you mention "This .store container must be located in root directory of the FWD-client" and "trusted-cert.store must exist in the root directory of the 'remote' client". In both these cases, by "root directory" do you mean "current" or "working" directory?

#53 - 09/10/2021 04:00 PM - Ovidiu Maxiniuc

Yes, the "working" directory is the right work. I have just updated the wiki.

#54 - 09/13/2021 12:32 PM - Roger Borrello

Ovidiu, I am trying to follow the wiki, and my server source code is the same as your example (except I am using a variable to hold the "20333c..." password). I get an error indicating the store has been tampered with.

I create the keystore:

```
rfb@rfb:~/testcases/deploy/server$ keytool -genkey -keyalg RSA -alias default_server -keystore default_server.
store -storepass password -validity 48 -keysize 1024
What is your first and last name?
  [Unknown]:  Roger Borrello
What is the name of your organizational unit?
  [Unknown]:  GCD
What is the name of your organization?
  [Unknown]:  Golden Code Development
What is the name of your City or Locality?
  [Unknown]:  Alpharetta
What is the name of your State or Province?
  [Unknown]:  GA
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Roger Borrello, OU=GCD, O=Golden Code Development, L=Alpharetta, ST=GA, C=US correct?
  [no]:  yes

Enter key password for <default_server>
    (RETURN if same as keystore password):

Warning:
The generated certificate uses a 1024-bit RSA key which is considered a security risk. This key size will be d
isabled in a future update.
The JKS keystore uses a proprietary format. It is recommended to migrate to PKCS12 which is an industry standa
rd format using "keytool -importkeystore -srckeystore default_server.store -destkeystore default_server.store
-deststoretype pkcs12".
```

Then I copied the store to the client directory: @cp default_server.store ../client/

I also added the password to the ./client.sh command line, so it comes through as: Dssl-socket:keystore:password=password

#55 - 09/13/2021 01:36 PM - Ovidiu Maxiniuc

That error I get when the password is somehow incorrect. Can you send me the default_server.store you've generated.

#56 - 09/13/2021 03:10 PM - Ovidiu Maxiniuc

Roger,
Your key loads in my test procedure(s). They are the exactly the same from Socket Handle and Server Socket Handle SSL Connections wiki. More than that, I am able to use firefox to connect to the 'server' and although the application is too dumb to be able to sustain a full HTTP request, the certificate is checked and displayed in web page.

The most likely cause for the error you get is the absence of the password for the .store file. Note that this is (or should be) a different password than the one for the key. The .store can store multiple keys, each having its alias/password. Please check if you have ssl-socket:keystore:password configured in the command line of the SSL-server endpoint client.

After the server procedure starts correctly (it should display SSL server started on port 9123. in message line), check the SSL-client endpoint client's command-line whether it as ssl-socket:truststore:password configured (if you generated the trusted-cert.store).

#57 - 09/14/2021 04:31 PM - Roger Borrello

This is what I did (still resulting in the tampered error).

```
rfb@rfb:~/testcases/deploy/server$ keytool -genkey -keyalg RSA -alias default_server -keystore default_server.
store -storepass keystore -validity 48 -keysize 1024     <-- "keystore"
What is your first and last name?
  [Unknown]:  Roger Borrello
What is the name of your organizational unit?
  [Unknown]:  GCD
What is the name of your organization?
  [Unknown]:  Golden Code
What is the name of your City or Locality?
  [Unknown]:  Alpharetta
What is the name of your State or Province?
  [Unknown]:  GA
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Roger Borrello, OU=GCD, O=Golden Code, L=Alpharetta, ST=GA, C=US correct?
  [no]:  yes

Enter key password for <default_server>
     (RETURN if same as keystore password):                                                    <-- "password"

Re-enter new password:
```

I then copied the default_server.store from ./deploy/server to ./deploy/client and now I run the server, and client (with -Dssl-socket:keystore:password=keystore) on the command line.

As you know, the code is using the "password" password.

#58 - 09/14/2021 04:52 PM - Ovidiu Maxiniuc

The problem is here:

> -Dssl-socket:keystore:password=keystore

In fact, there are two of them:

1. use password password, not keystore;
2. this is not a property to define, but a parameter of the ClientDriver Java app. Drop the -D and move it after the class name, similar to net:server:insecure_port=3333.

#59 - 09/14/2021 05:39 PM - Roger Borrello

OK... because I am having issues with the below, I'll flip-flop the passwords. I get Unable to prtocess parameters. (5509) and Invalid parameter string. (5510).

```
define variable kapw as char init "keyword".
define variable kapw_encrypted as char.

hsSocket:SET-CONNECT-PROCEDURE("handle-connection").
assign kapw_encrypted = ENCRYPT(kapw).
hsSocket:ENABLE-CONNECTIONS("-S 9123 -ssl -keyaliaspasswd " + kapw_encrypted).
```

#60 - 09/15/2021 09:33 AM - Ovidiu Maxiniuc

I understand now. And I have just noticed the "keystore" password commented in your note-57.

The problem is that you cannot use ENCRYPT function to encrypt the password. This method outputs a very different result than expected. I did not test, but I think your code does not work on 4GL.

In fact, 4GL does not offer a facility to allow the user to enter the password in a program and programmatically convert it to a form usable in ENABLE-CONNECTION method. Instead, the password must be encoded externally using the provided genpassword utility. (Of course this utility cane be executed as an external application and the result read from its output, but this is a workaround.) The FWD replacement for this utility is SymmetricEncryption.encrypt() static method. Note that this is different than SecurityOps.encode() methods which are the counterpart for 4GL's ENCODE and SecurityOps.encrypt() which mirrors the ENCRYPT function (and which returns a MEMPTR, not a CHARACTER).

Refer to this page (and the reset of the related documents) for more informations:
https://docs.progress.com/bundle/openedge-security-keys-and-certificates-122/page/Use-genpassword-to-obtain-a-keystore-password-encrypted-value.html

#61 - 09/15/2021 02:00 PM - Roger Borrello

Ovidiu, thank you for all those pointers. Thanks to them, I was successful in making sure the store password and the alias password! (Now I am working on the connectivity itself).

I think that the wiki should be a little more step-by-step with the example, including basic passwords for the store and the alias, and more of the order in which they should be performed.

#62 - 09/15/2021 02:11 PM - Greg Shah

> I think that the wiki should be a little more step-by-step with the example, including basic passwords for the store and the alias, and more of the order in which they should be performed.

I agree.

Roger: Could you please use your insight gained from this process to make the first pass that these changes? Then Ovidiu will review and correct anything as needed.

#63 - 09/15/2021 02:24 PM - Ovidiu Maxiniuc

Roger Borrello wrote:

> Ovidiu, thank you for all those pointers. Thanks to them, I was successful in making sure the store password and the alias password! (Now I am working on the connectivity itself).

You're welcome!

> I think that the wiki should be a little more step-by-step with the example, including basic passwords for the store and the alias, and more of the order in which they should be performed.

OK, I will update the wiki according to your hints.

#64 - 09/15/2021 02:36 PM - Roger Borrello

I am troubled by the fact that I can use my current configuration with the sockets/server-connect.p and sockets/client-socket.p successfully, but I cannot with the sockets/ssl/ssl-server.p and sockets/ssl/ssl-client.p.

My server is a GCD laptop (192.168.40.165) and the client is an Ubuntu VM on my other GCD laptop (192.168.40.226).

Like I said, the basic testcases connect fine. The server issues serverSocket:ENABLE-CONNECTIONS("-S 3363"). and the client issues clientSocket:CONNECT ("-H 192.168.40.165 -S 3363"). The ssl testcases are almost exactly like in the wiki. I even wiped out all the IPTABLES on the GCD laptop, in case the firewall was creating a problem.

What is the basic network setup you have used to successfully make the connection?

#65 - 09/15/2021 03:36 PM - Ovidiu Maxiniuc

I run the tests only from my workstation. No network changes required. The parameters look fine, I am positive the firewall is the one which prevent the client to connect to the server. You can try opening a browser on .165 and go to http://localhost:3363 or even use telnet for unsecured connection. To see SSL in action use https://localhost:3363 in the browser.

To test locally (on a single workstation) you can either:

- use two FWD server instances running on two different base ports (a bit complex to configure);
- use uast/ask.p test procedure from testcases repository - this is what I use because this is the simplest to configure and grants full binary compatibility.

However, if the plain (unsecured) variant works, there is not a networking issue. I think the SSL handshake failed because the client could not validate the server. Make sure that either:

- the -nohostverify option is specified in client's parameter
- the certificate for the server exists (trusted-cert.store) and is accessible to client process (including the access password as command line parameter).

#66 - 09/15/2021 05:36 PM - Roger Borrello

Well it was just a matter of time before I figured out what I was doing wrong. I hadn't noticed that there were 2 separate steps to creating the keystore/trust configuration, and I was using the same ssl-socket option to both the socket creator (server) and the socket connector (client). It was the use of "FWD-client" that confused me. In any case, I think the wiki does need a little bit of reorganization, such that a "knuckle-head" like me can follow it :-). I'll need some more information.

There is a pre-configuration step to perform, outside of the programming to generate:

- the keystore for the server
- a trusted certificate for the client

How would one create a keystore without an access password? Just not pass any password to the keytool?

There are 3 files involved. In our example:

- default_server.store
- default_server.cer

- trusted-cert.store

Is the .cer file just an interim file that isn't needed during execution? Then there's the item about where to locate them. For the sake of discussion, let's say I have deploy_server and deploy_client, both with the server and client directories within. Which would require which files? And does the value in the clientConfig/workingDir section of the directory control any of that?

So I can update the wiki, but I'll just need a bit more information to get it right.

#67 - 09/15/2021 06:04 PM - Roger Borrello

- File 4366_tls_and_ssl.png added

I can finally accomplish the testing. I wanted to check that when the server is setup for ssl and the client is tls, we can still connect, but the data is garbled? Or should we not be able to make the connection?

#68 - 09/15/2021 06:47 PM - Roger Borrello

I guess there are a limited number of permutations to deal with. Below is my testplan:

Server	Client	Result
-ssl	-ssl	Success
-sslv2	-sslv2	TBD
-sslv3	-sslv3	TBD
-tls	-tls	Success
-tlsv1	-tlsv1	TBD
-tlsv1.1	-tlsv1.1	TBD
-tlsv1.2	-tlsv1.2	TBD
-ssl	-tls	TBD
-sslv2	-tls	TBD
-sslv3	-tls	TBD
-tls	-ssl	TBD
-tls	-sslv2	TBD
-tls	-sslv3	TBD
-tlsv1	-ssl	TBD
-tlsv1	-sslv2	TBD
-tlsv1	-sslv3	TBD
-tlsv1.1	-ssl	TBD
-tlsv1.1	-sslv2	TBD
-tlsv1.1	-sslv3	TBD
-tlsv1.2	-ssl	TBD
-tlsv1.2	-sslv2	TBD
-tlsv1.2	-sslv3	TBD
-ssl	-tlsv1	TBD
-ssl	-tlsv1.1	TBD
-ssl	-tlsv1.2	TBD
-sslv2	-tlsv1	TBD
-sslv2	-tlsv1.1	TBD
-sslv2	-tlsv1.2	TBD
-sslv3	-tlsv1	TBD
-sslv3	-tlsv1.1	TBD
-sslv3	-tlsv1.2	TBD

That's a hefty number of edge cases.

#69 - 09/16/2021 08:09 AM - Greg Shah

> I wanted to check that when the server is setup for ssl and the client is tls, we can still connect, but the data is garbled?

Not necessarily. The SSL and TLS standards often have some kind of negotiated fallback to a common supported approach. This happens during the protocol "handshake". For example, TLS 1.2 can fallback to SSL 3.0 but cannot fallback to not SSL 2.0. For example, testing the -sslv2 with -tls should not work since TLS 1.2 can only fall back to SSL 3.0 and SSL 2.0 can't support SSL 3.0.

I think we can focus on testing the server side for -tls which is our more important use case. We could do that via a web browser as a client. The down side there is that the older protocol versions can't be tested with the latest browsers because the protocol support was removed. If you can use a recent browser version to connect to the server socket and the connection reports TLS 1.2, then I think we are OK. To do this, you need to write a little 4GL code to act like a web server.

A web server will listen for a new connection. When the connection occurs, it will read a line from the socket. That line will be in the form of <verb> <uri> <http-protocol><newline>. There will be additional lines with the rest of the request but these don't matter for a simple GET. An example: GET /index.html HTTP/1.1<newline>. Notice that the entire URL is not there because the host/port portions are not part of the request, they are used to create the connection. The web server then writes a response in the form:

HTTP/1.1 200 OK<newline>Server: Braindead Web Server<newline>Date: Thu, 16 Sep 2021 07:56:07 GMT<newline>Content-type: text/html<newline>Content-length: 45<newline><newline><!DOCTYPE html><html><body>Test</body></html>

At that point the web server closes the socket. This implementation doesn't do anything useful other than return the same "page" over and over, no matter what the request is. The browser can tell you about the connection. In the browser, you'll have to bypass the warning and accept the certificate since you are using self-signed certs.

#70 - 09/16/2021 08:50 AM - Ovidiu Maxiniuc

Roger Borrello wrote:

> There is a pre-configuration step to perform, outside of the programming to generate:
>
> - the keystore for the server
> - a trusted certificate for the client

True. But these are customer's resources. We create these just to see the code works. Normally, a production setup will not work with self-generated/signed key/certificate. Alternatively, the customer may chose to reuse the old certificate/keys, in which case he will import them into JSK format.

> How would one create a keystore without an access password? Just not pass any password to the keytool?

I have just tested it. You cannot. It was a false memory. Sorry. Also There is a misinformation in the wiki about the ENCRYPT() function. As noted above, the genpassword (P4GL) / SymmetricEncryption.encrypt() (FWD) must be used instead.

> There are 3 files involved. In our example:
>
> - default_server.store
> - default_server.cer
> - trusted-cert.store
>
> Is the .cer file just an interim file that isn't needed during execution?

Yes. That is correct. The .cer is not needed at runtime, but can be used for sharing the certificate if other applications intend to connect to out SSL server. This format is more common than JKS.

> Then there's the item about where to locate them. For the sake of discussion, let's say I have deploy_server and deploy_client, both with the server and client directories within. Which would require which files? And does the value in the clientConfig/workingDir section of the directory control any of that?

	deploy_server		deploy_client	
	Server	Client	Server	Client
Resource file (in working directory)	*none*	default_server.store	*none*	trusted-cert.store
Command-line Parameter	*none*	ssl-socket:keystore:password	*none*	ssl-socket:truststore:password

#71 - 09/16/2021 09:02 AM - Ovidiu Maxiniuc

Roger Borrello wrote:

> I guess there are a limited number of permutations to deal with. Below is my testplan:

Why not re-arrange the table as:

		Server						
		-ssl	-sslv2	-sslv3	-tls	-tlsv1	-tlsv1.1	-tlsv1.2
Client	-ssl	Success	TBD	TBD	TBD	TBD	TBD	TBD

	-sslv2	TBD	TBD	TBD	TBD	TBD	TBD	TBD
	-sslv3	TBD	TBD	TBD	TBD	TBD	TBD	TBD
	-tls	TBD	TBD	TBD	Success	TBD	TBD	TBD
	-tlsv1	TBD	TBD	TBD	TBD	TBD	TBD	TBD
	-tlsv1.1	TBD	TBD	TBD	TBD	TBD	TBD	TBD
	-tlsv1.2	TBD	TBD	TBD	TBD	TBD	TBD	TBD

#72 - 09/16/2021 10:27 AM - Roger Borrello

Ovidiu Maxiniuc wrote:

> Why not re-arrange the table as:

Because I am unworthy! Awesome!

Let me follow-up with a question. If I have a customer's default_sever.pem file from another OE implementation, how would I incorporate that into a new configuration. In this case, it's a different OE client, and a FWD server. But it could be a FWD client, as well.

#73 - 09/16/2021 11:42 AM - Ovidiu Maxiniuc

Roger Borrello wrote:

> Because I am unworthy! Awesome!

Haha □□!

> Let me follow-up with a question. If I have a customer's default_sever.pem file from another OE implementation, how would I incorporate that into a new configuration. In this case, it's a different OE client, and a FWD server. But it could be a FWD client, as well.

I never did this. I understand that this is a 2 step process, using the intermediary universal pkcs12 format and I guess the commands are:

```
openssl pkcs12 -export -in default_sever.pem -out default_sever.pkcs12
keytool -importkeystore -srckeystore default_sever.pkcs12 -srcstoretype pkcs12 -destkeystore default_sever.sto
re
```

#74 - 09/16/2021 01:35 PM - Roger Borrello

Below are my findings... I hope the table looks OK.

		Server						
		-ssl	-sslv2	-sslv3	-tls	-tlsv1	-tlsv1.1	-tlsv1.2
Client	-ssl	Success	SSL fail (1)	SSL fail (1)	SSL fail (1)	SSL fail (1)	SSL fail (1)	SSL fail (1)
	-sslv2	Connect (2)	Success	Success	Success	Success	Success	Success
	-sslv3	Connect (2)	Success	Success	Success	Success	Success	Success
	-tls	Connect (2)	Success	Success	Success	Success	Success	Success
	-tlsv1	Connect (2)	Success	Success	Success	Success	Success	Success
	-tlsv1.1	Connect (2)	Success	Success	Success	Success	Success	Success
	-tlsv1.2	Connect (2)	Success	Success	Success	Success	Success	Success

(1) SSL failure -54 / unable to get local issuer certificate for a4b3c2d1.0 in trusted-cert.store (9318) / Garbage data on server
(2) Client indicated connection made, but not server. Garbled data returned.

#75 - 09/16/2021 06:26 PM - Roger Borrello

What other variety of tests would be helpful? Do these results coincide with expected results? To me it looks like negotiation is a little confusing.

#76 - 09/16/2021 07:16 PM - Greg Shah

- Status changed from Test to Closed

Files

4366_tls_and_ssl.png	8.94 KB	09/15/2021	Roger Borrello