Using IceSSL

Incorporating IceSSL into your application requires installing the plug-in, configuring it according to your security requirements, and creating SSL endpoints.

On this page:

- Installing IceSSL
  - C++ Applications
  - Java Applications
  - .NET Applications
  - Ice Touch Applications
- Creating SSL Endpoints
- Endpoint Security Considerations

Installing IceSSL

Ice supports a generic plug-in facility that allows extensions (such as IceSSL) to be installed dynamically without changing the application source code. The `Ice.Plugin` property provides language-specific information that enables the Ice run time to install a plug-in.

C++ Applications

The executable code for the IceSSL C++ plug-in resides in a shared library on Unix and a dynamic link library (DLL) on Windows. The format for the `Ice.Plugin` property is shown below:

```
Ice.Plugin.IceSSL=IceSSL:createIceSSL
```

The last component of the property name (`IceSSL`) becomes the plug-in's official identifier for configuration purposes, but the IceSSL plug-in requires its identifier to be `IceSSL`. The property value `IceSSL:createIceSSL` is sufficient to allow the Ice run time to locate the IceSSL library (on both Unix and Windows) and initialize the plug-in. The only requirement is that the library reside in a directory that appears in the shared library path (`LD_LIBRARY_PATH` on most Unix platforms, `PATH` on Windows).

Additional configuration properties are usually necessary as well.

Java Applications

The format for the `Ice.Plugin` property is shown below:

```
Ice.Plugin.IceSSL=IceSSL.PluginFactory
```

The last component of the property name (`IceSSL`) becomes the plug-in's official identifier for configuration purposes, but the IceSSL plug-in requires its identifier to be `IceSSL`. The property value `IceSSL.PluginFactory` is the name of a class that allows the Ice run time to initialize the plug-in. The IceSSL classes are included in `Ice.jar`, therefore no other changes to your `CLASSPATH` are necessary.

Additional configuration properties are usually necessary as well.

.NET Applications

The format for the `Ice.Plugin` property is shown below:

```
Ice.Plugin.IceSSL=C:/Ice/bin/IceSSL.dll:IceSSL.PluginFactory
```

The last component of the property name (`IceSSL`) becomes the plug-in's official identifier for configuration purposes, but the IceSSL plug-in requires its identifier to be `IceSSL`. The property value contains the file name of the IceSSL assembly as well as the name of a class, `IceSSL.PluginFactory`, that allows the Ice run time to initialize the plug-in.

You may also specify a partially or fully qualified assembly name instead of the file name in an `Ice.Plugin` property. For example, you can use the following configuration to load the plug-in from the release version of the assembly provided in ZeroC's binary distribution:

Release Key

```
Ice.Plugin.IceSSL=IceSSL,Version=3.4.2.0,Culture=neutral,PublicKeyToken=cdd571ade22f2f16:IceSSL.
PluginFactory
```

Alternatively, if you have built the Ice source code yourself, you can load the development version of the assembly as shown below:

Development Key

```
Ice.Plugin.IceSSL=IceSSL,Version=3.4.2.0,Culture=neutral,PublicKeyToken=1f998c50fec78381:IceSSL.
PluginFactory
```

You *must* use a fully qualified assembly name to load the IceSSL plug-in from the Global Assembly Cache.

Additional configuration properties are usually necessary as well.

Ice Touch Applications

The IceSSL plug-in is included in the Ice Touch run time and installed automatically, therefore it is not necessary to explicitly load it. However, configuration properties are usually necessary.

Creating SSL Endpoints

Installing the IceSSL plug-in enables you to use a new protocol, `ssl`, in your endpoints. For example, the following adapter endpoint list creates a TCP endpoint, an SSL endpoint, and a UDP endpoint:

```
MyAdapter.Endpoints=tcp -p 4063:ssl -p 4064:udp -p 4063
```

As this example demonstrates, it is possible for a UDP endpoint to use the same port number as a TCP or SSL endpoint, because UDP is a different protocol and therefore has an indepdendent set of ports. It is not possible for a TCP endpoint and an SSL endpoint to use the same port number, because SSL is essentially a layer over TCP.

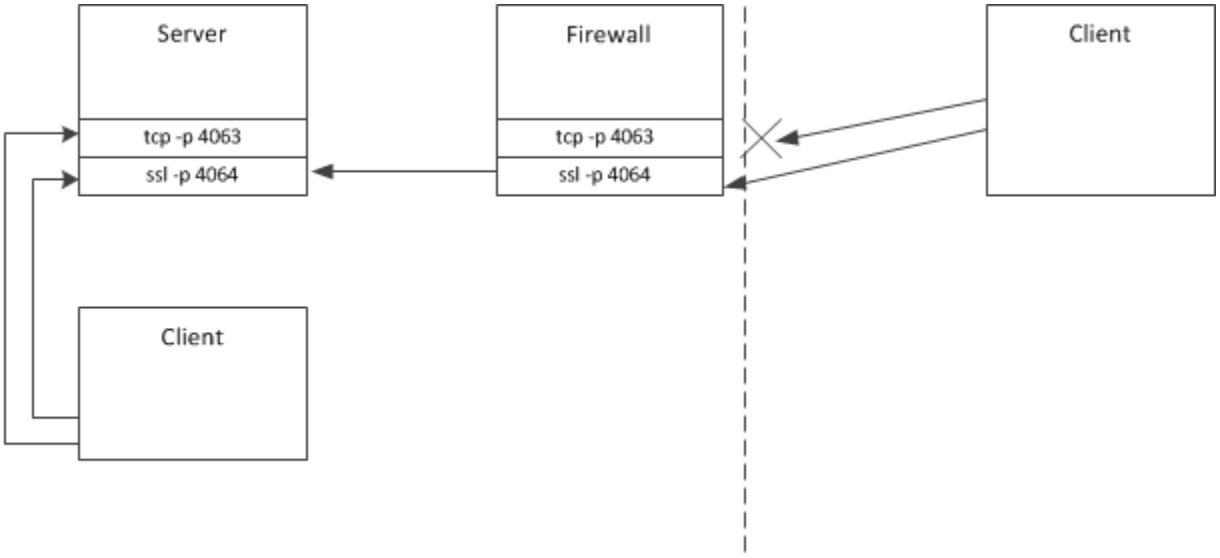
Using SSL in proxy endpoints is equally straightforward:

```
MyProxy=MyObject:tcp -p 4063:ssl -p 4064:udp -p 4063
```

Endpoint Security Considerations

Defining an object adapter's endpoints to use multiple protocols, as shown above, has obvious security implications. If your intent is to use SSL to protect session traffic and/or restrict access to the server, then you should only define SSL endpoints.

There can be situations, however, in which insecure endpoint protocols are advantageous. The figure below illustrates an environment in which TCP endpoints are allowed behind the firewall, but external clients are required to use SSL:



An application of multiple protocol endpoints.

The firewall in the illustration is configured to block external access to TCP port 4063 and to forward connections to port 4064 to the server machine.

One reason for using TCP behind the firewall is that it is more efficient than SSL and requires less administrative work. Of course, this scenario assumes that internal clients can be trusted, which is not true in many environments.

For more information on using SSL in complex network architectures, refer to our discussion of the Glacier2 router.

See Also

- Plug-in Facility
- Configuring IceSSL
- Object Adapter Endpoints
- Proxy Endpoints
- Glacier2
- Ice Plug-In Properties