这部分主要涉及企业级应用的安全问题，一般来说安全框架主要提供3个典型的安全行为：认证、授权和审核。除了典型的安全问题，对于一个以消息作为通信手段的分布式应用，还需要考虑消息保护(Message Protection)问题，消息保护机制主要包括签名和加密，前者保证消息的一致性，后者保证消息的机密性。

在介绍传输安全之前，介绍几个在分布式应用中常见的传输安全隐患：消息的篡改，敏感信息的泄露，钓鱼攻击，重放攻击。因此对于WCF来说，其传输安全主要涉及认证、消息一致性和机密性三个主题，认证不仅包括服务器对客户端的认证，也包括客户端对服务的身份验证，即双向验证，消息一致性保证消息的内容在传输过程中不被篡改，机密性确保只有希望的消息接收方才能读取其中内容。WCF为了应对这些问题，提供了两种不同的安全模式，Transport安全和Message安全。

**Transport安全**：基于传输层协议的安全机制，其中TLS/SSL是最常用的方式，常说的HTTPS其实就是将HTTP和TLS/SSL结合在一起，对于WCF来说，所有的基于HTTP协议的绑定在采用Transport安全的情况，对于NetTcpBinding来说，也同样支持，即组合使用TCP和TLS/SSL。该协议体系可以解决如下两个问题：客户端对服务端的验证；通过对传输层传输的数据段进行加密确保消息的机密性。接下来通过一个例子，来描述**连接HTTPS的过程**。

步骤1：客户端向HTTPS站点发送协商请求，该请求中包含客户端所能够支持的加密算法列表。

步骤2：HTTPS站点从加密算法列表中选择自己支持的并且安全级别最高的算法，连同绑定到该站点的数字证书(X.509证书)一并发个客户端。

步骤3：客户端接受到站点发回的数字证书后，通过验证证书进而确定站点身份，在验证成功的情况下，客户端会生成一个随机数，作为会话密钥(Session Key)，缓存在客户端。客户端会采用站点发回的加密算法，利用从证书中提取的公钥进行加密。加密后的会话密钥发送给站点后，站点使用自己的私钥解密，至此客户端和服务端具有一个只有彼此知晓的**会话密钥**，所有请求消息和回复消息均用其加解密。（由于非对称加密和数字签名的知识相对基础，就不详细介绍了，如果需要，可以参见<http://www.cnblogs.com/wanliwang01/p/aspnet_webapi_base01.html>。

Transport安全模型的最大的优点就是高性能，虽然在消息交换前需要一个协商的过程，不过可以通过硬件加速。其不足是：依赖于集体的传输协议；只能提供点到点的安全，即客户端直接连接到服务端的场景，如果需要增加消息路由的中间节点，也无法使用了；如果选择该模型，意味着需要在传输层解决对客户端的认证，但相应方案较少。

**Message安全**：直接将安全策略的目标对象转移到消息本身，通过对消息进行签名、加密实现消息安全传输。其不依赖与具体的协议，并可以提供端到端的安全，其是一种应用层的协议，配套方案很多。WCF的Message安全模式是围绕4个标准的WS-\*规范建立的，包括WS-Security、WS-Trust、WS-Secure Conversation和WS-Security Policy。

**Mixed安全**：由于前两者都有着自己的优点和缺点，因此综合考虑，存在如下的解决方案：消息的一致性、机密性和客户端对服务端的认证通过Transport安全模式实现，而服务端对客户端的认证采用Message安全模式。

**认证和凭证：**常见的认证方式包括用户名密码认证，例如Windows认证、Membership模块、自定义认证等；NTLM，windows认证是实现单点登录的理想方式，其通过账号密码得到一个凭证，在凭证超时前，可以用于任何应用；Kerberos，其比NTLM更加高效，安全，涉及客户端、服务端和密钥分发中心3方，整个过程包括获得“认购权证“、通过”认购权证“购买”入场券“、凭票入场；数字证书认证，采用信任链的方式实现。

之前介绍的主要是安全概念，接下来则在WCF中，安全的具体实践。以最简单的BasicHttpBinding为例，其SecurityMode包括None、Transport、Message、TransportWithMessageCredential（等价Mixed）和TransportCredential。对于其中的Transport安全来说，其又包含6中客户端凭证类型，None、Basic、Digest、Ntlm、Windows、Certificate。比如如下两个基于basicHttpBinding配置，前者基于Message模式+X.509证书凭证，后者采用Mixed模式+用户名/密码凭证。其他的绑定类型虽然有不少差异，但原理一致，在这就不一一展开了。

|  |
| --- |
| <system.serviceModel>  <bindings>  <basicHttpBinding>  <binding name="messageBinding">  <security mode="Message">  <message clientCredentialType="Certificate" **negotiateServiceCredential="false"**/>  </security>  </binding>  <binding name="transportWithMessageCredentialBinding">  <security mode="TransportWithMessageCredential">  <message clientCredentialType="UserName"/>  </security>  </binding>  </basicHttpBinding>  </bindings>  </system.serviceModel> |

Tip:可以通过设置message的协商属性来决定是否需要协商。

之前也曾提到，对于认证来说包括服务端认证和客户端认证，涉及ServiceCredentials和ClientCredentials两大类凭证。首先介绍服务认证，常见的服务端认证配置如下所示。

|  |
| --- |
| <system.serviceModel>  <services>  <service name="Sory.CoreFramework.Service.EmployeeService" behaviorConfiguration="serviceCertificateBehavior"></service>  </services>  <behaviors>  <serviceBehaviors>  <behavior name ="serviceCertificateBehavior">  <serviceCredentials>  <serviceCertificate storeLocation="LocalMachine" storeName="My" x509FindType="FindBySubjectName" findValue="Xionger-PC"/>  </serviceCredentials>  </behavior>  </serviceBehaviors>  </behaviors>  </system.serviceModel> |

通过MakeCert.exe工具创建一个证书

|  |
| --- |
| Makecert –n “CN=RootCA” –r –sv C:\Root.pvk C:\RootCA.cer  Makecert –n “CN=Xionger-PC” –ic C:\RootCA.cer –iv C:\RootCA.pvk –sr  LocalMachine –ss My –pe –sky exchange |

在WCF中，服务身份通过ServiceEndpoint表示，在Windows认证下，通常使用SPN(Service Principal Name)和UPN（User Principal Name）两种，如果采用X.509证书，可以通过X509CertificateEndpointIdentity和RsaEndpointIdentity表示。在服务引用或使用SvcUtil.exe导入元数据时，会将服务身份标识自动写入配置中，如下所示。

|  |
| --- |
| <system.serviceModel>  <client>  <endpoint>  <identity>  <userPrincipalName value="xionger@126.com"/>  <servicePrincipalName value="host/xionger-PC"/>  <certificateReference storeLocation="LocalMachine" storeName="My" x509FindType="FindBySubjectName" findValue="xionger-PC"/>  </identity>  </endpoint>  </client>  </system.serviceModel> |

客户端认证主要包含三种方式：Windows，其基于SSPI，这部分曾经在连接字符串中见到；用户名，支持Windows、Membership、Custom三种，尤其是Membership的使用，请见接下来的代码示例；X.509证书，在客户端可以通过ChannelFactory.Credentials.ClientCertificate.SetCertificate方法设置，此外证书与Windows账号映射可以通过<clientCertificate>节中<authentication mapClientCertificateToWindowsAccount=”true”>设置。

|  |
| --- |
| 客户端认证，用户名方式的MembershipProvider  <system.web>  <membership defaultProvider="myProvider">  <providers>  <add name="myProvider" type="System.Web.Security.SqlMembershipProvider, System.Web" connectionStringName="AspNetDb"  applicationName="MembershipAuthenticationDemo" requiresQuestionAndAnswer="false"/>  </providers>  </membership>  </system.web>  <system.serviceModel>  <behaviors>  <serviceBehaviors>  <behavior name="membershipAuthentication">  <serviceCredentials>  <serviceCertificate/>  <userNameAuthentication userNamePasswordValidationMode="MembershipProvider" membershipProviderName="myProvider"/>  </serviceCredentials>  </behavior>  </serviceBehaviors>  </behaviors>  </system.serviceModel> |

**消息保护**，这部分保证消息的一致性和机密性，在WCF中，通过消息保护级别的概念来设置，包括None、Sign和EncryptAndSign三个级别，默认为EncryptAndSign级别，这部分的功能是通过之前章节介绍过的SecurityBindingElement的相关类来实现的。此外，为了减少多次认证的开销，还有一个关于安全会话的概念，通过配置Binding->security->message中的establishSecurityContext属性来实现，可以使多次消息交换使用同一个会话信道，这部分内容很多，还需要足够的实践。不过不管是什么平台和技术，基本的安全概念是相似的，在传输过程中，就是认证、数据一致性和机密性。

在介绍完认证Authentication后，就进入了授权的模块，当然还包含所有安全过程的审核工作。对于整个.NET体系来说，其用户和角色等信息都是通过身份Identity和安全主体Principal两个概念来表述的。前者包含用户的基本令牌信息，可以是WindowsIdentity、GenericIdentity和X509三种类型，这儿值得一提的是GenericIdentity，如果采用自定义认证方式时，会选择GenericIdentity这个类型。在服务安全开启的情况下，服务端在经过认证后会创建一个上下文用于存储基于当前服务调用相关的安全相关的信息，其关系如下表所示。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| None | Windows | UserName | | | Certificate | |
| windows | Membership | Custom | Default | Windows Account Mapping |
| Generic Identity | Windows Identity | Windows Identity | Generic Identity | Generic Identity | X509 Identity | Windows Identity |

接下来，介绍主体的概念，简单来说，主体是对标识的封装，增加了所属角色这一关联属性。在windows中，安全主体被保存在TLS线程本地变量上，可以通过Thread.CurrentPrincipal获取。

常见授权方式包括Windows用户组授权、ASP.NET Roles授权和自定义授权方式三种。Windows授权相对简单，设置behavior.PrincipalPermissionMode= PrincipalPermissionMode.UseWindowsGroups即可，通过在方法上添加[PrincipalPermission(SecurityAction.Demand, Role=”administrators”)]控制windows下用户的操作，这部分还涉及一个Impersonation身份模拟的概念，和win32有关比较复杂，就不细致介绍了。对于ASP.NET Roles提供程序来说，System.Web.Security.RoleProvider抽象类是其基础，主要使用其子类SqlRoleProvider来处理，其配置如下所示。

|  |
| --- |
| <connectionStrings>  <add name="aspNetDb" connectionString="..." providerName="System.Data.SqlClient"/>  </connectionStrings>  <system.web>  <roleManager enabled="true" defaultProvider="SqlRoleProvider">  <providers>  <add name="sqlRoleProvider" type="System.Web.Security.SqlRoleProvider, System.Web" connectionStringName="aspNetDb" applicationName="AspRolesAuthorizationDemo"/>  </providers>  </roleManager>  </system.web>  <system.serviceModel>  <services>  <service behaviorConfiguration="aspNetRolesAuthorization"></service>  </services>  <behaviors>  <serviceBehaviors>  <behavior name="aspNetRolesAuthorization">  <serviceAuthorization principalPermissionMode="UseAspNetRoles" roleProviderName="sqlRoleProvider"></serviceAuthorization>  </behavior>  </serviceBehaviors>  </behaviors>  </system.serviceModel> |

此外，还有自定义的授权方式，其设计AuthorizationPolicy、ServiceAuthorization、Claim和ClaimSet等多个类型，实现很复杂，需要时再查书实践。

最后介绍安全审核部分，这部分其实和windows的事件管理器关系非常紧密，最简单的配置就是在behavior节中，设置<serviceSecurityAudit auditLogLocation=”Application” messageAuthenticationAuditLevel=”SuccessOrFailure”>。

虽然是这样的浏览学习，不过也算是圆梦之旅，棒棒哒。

参考资料：

[1]蒋金楠. WCF全面解析[M]. 上海:电子工业出版社, 2012.