Enoncé complémentaire 4

Tips :

1. Implémenter un système d'autorisation avec WCF

1.1. Création d'un service WCF

Créer un service WCF comme expliqué dans l'énoncé complémentaire 2.

1.2. Configuration du binding

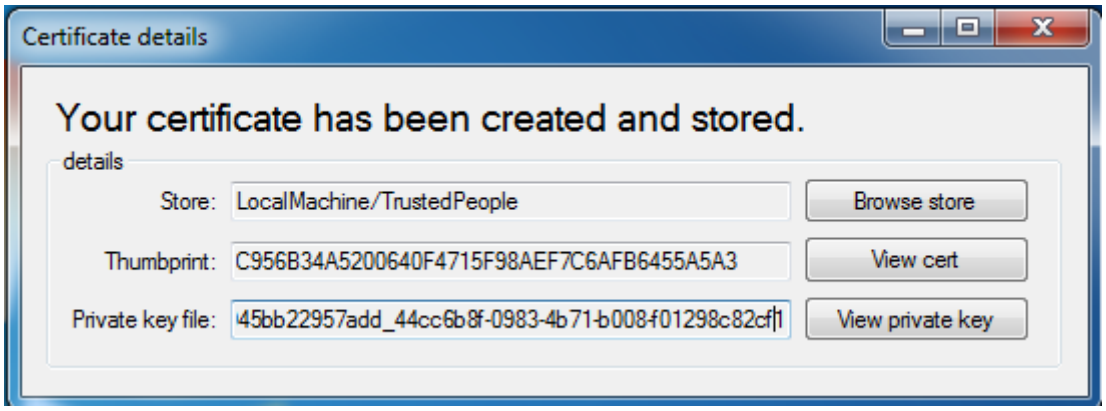
Nous allons ajouter un wsHttpBinding à notre service. Le wsHttpBinding permet une liaison fiable et sécurisé sur le protocole http. Nous définissons le type « Message » pour le mode de sécurité. Cela signifie que la sécurité sera appliquée au niveau du message transféré (il sera donc encrypté). Cette gestion de la sécurité est donc indépendante du protocole utilisé. On choisit UserName pour le type de credentials. Cela signifie que l'authentification se fera sur un couple username/password classique géré au niveau du serveur. On rajoute donc dans la balise system.servicemodel de notre fichier Web.config le code ci-dessous :

```
<protocolMapping>
    <add scheme="http" binding="wsHttpBinding" />
</protocolMapping>
<bindings>
    <wsHttpBinding>
       <binding>
         <security mode="Message">
           <message clientCredentialType="UserName"/>
         </security>
       </binding>
    </wsHttpBinding>
</bindings>
```

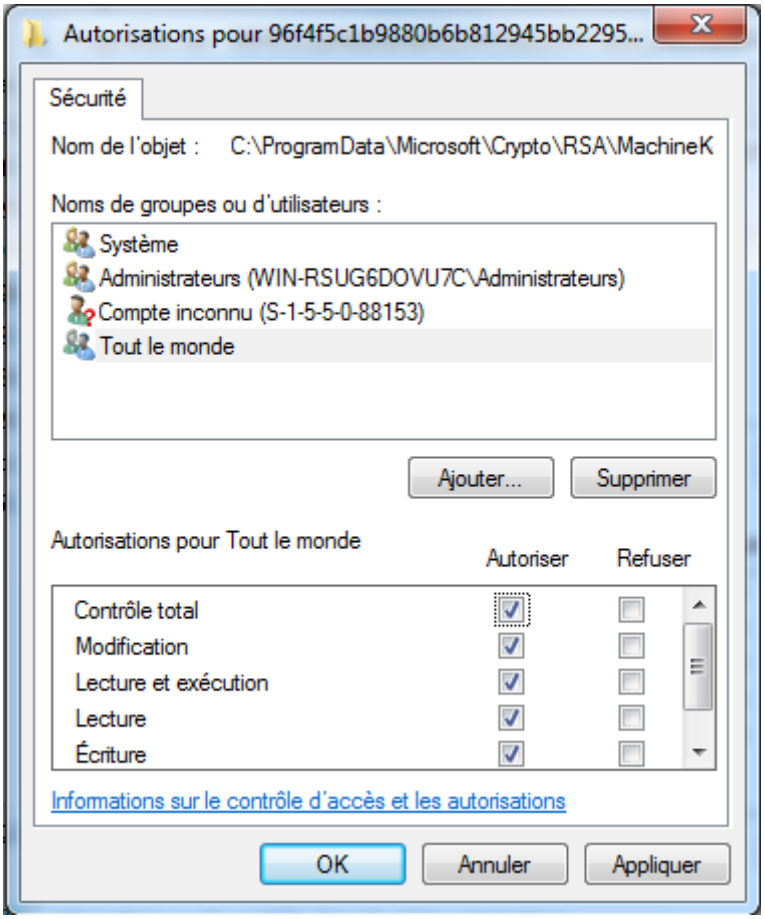
1.3. Installation du certificat

Il est nécessaire d'installer un certificat sur le serveur afin que les données transférées soient signées et que le client puisse être sûr que les informations qu'il reçoit proviennent du serveur. Pour les tests, nous allons donc créer et installer un certificat de test. Ces étapes laborieuses peuvent être réalisées en un clic directement à l'aide de l'outil SelfCert (https://s3.amazonaws.com/pluralsight-free/keith-brown/samples/SelfCert.zip). Décompresser l'archive et exécuter l'outil en mode administrateur :

On spécifie le nom du certificat précédé par « cn= ». On choisit aussi de stocker notre certificat dans le magasin « TrustedPeople », ce qui signifie que notre client considérera que ce certificat est sûr sans avoir besoin de le valider. On clique sur « Save » afin de générer et installer notre certificat de Test.



Dans la fenêtre ci-dessus, on récupère le chemin du fichier contenant la clé privée et on utilise l'explorateur Windows afin d'accorder toutes les autorisations sur cette clé :



1.4. Ajout du certificat dans la configuration du serveur et gestion de l'authentification

On rajoute le code suivant dans la balise behavior du fichier Web.config :

```
<serviceCredentials>
       <serviceCertificate findValue="CertificatTest"
        storeLocation="LocalMachine" storeName="TrustedPeople"
        x509FindType="FindBySubjectName"/>
       <userNameAuthentication userNamePasswordValidationMode="Custom"
        customUserNamePasswordValidatorType="WCFSecurite.CustomUserNameValidator,
        WCFSecurite"/>
</serviceCredentials>
```

Dans ce code, nous avons donc défini le nom de notre certificat ainsi que le magasin où il est stocké. Nous avons aussi spécifié que la validation de l'authentification se fait dans une classe personnalisée CustomUserNameValidator qu'il faut définir :

```
using System;
using System.IdentityModel.Selectors;

namespace WCFSecurite
{
    public class CustomUserNameValidator : UserNamePasswordValidator
    {
        public override void Validate(String userName, String password)
        {
            if (userName == null || password == null)
            {
                throw new ArgumentNullException();
            }

            if (!XXXXXX.verifieUtilisateur(userName, password))
            {
                throw new Exception("Utilisateur ou Mot de passe incorrect");
            }
        }
    }
}
```

La classe XXXXXX peut par exemple définir un accès à une base de données où seraient stockés les couples username/password de notre application. Le username du client ayant fait appel au service peut se récupérer dans le code côté serveur de la manière suivante :

```
OperationContext oc = OperationContext.Current;
ServiceSecurityContext ssc = oc.ServiceSecurityContext;
String username = ssc.PrimaryIdentity.Name;
```

2. Autorisations par les rôles

On va définir ici des autorisations par les rôles au niveau des méthodes de notre service :

```
using System;
using System.Runtime.Serialization;
using System.Security.Principal;
using System.Security.Permissions;
using System.ServiceModel;
using System.Text;

namespace WCFSecurite
{
    public class WCFService : IWCFService
    {
        [PrincipalPermission(SecurityAction.Demand, Role = "user")]
        public String getInfoPublique()
        {
            return "Info publique";
        }

        [PrincipalPermission(SecurityAction.Demand, Role = "VIP")]
        public String getInfoLouche()
        {
            return "Info louche";
        }
    }
}
```

Dans notre exemple, la méthode getInfoPublique ne pourra être exécutée que par des clients qui appartiennent aux rôles user alors que la méthode getInfoLouche ne pourra elle qu'être executée par des clients qui appartiennent aux rôles VIP. Pour gérer les autorisations par les rôles, on rajoute le code suivant dans la balise behavior du fichier Web.config :

```
<serviceAuthorization principalPermissionMode="UseAspNetRoles"
 roleProviderName="CustomProvider" />
```

Dans la balise system.web du fichier Web.config, on ajoute :

```
<roleManager enabled="true" defaultProvider="CustomProvider">
  <providers>
    <add name="CustomProvider" type="WCFSecurite.CustomRoleProvider, WCFSecurite" />
  </providers>
</roleManager>
```

Dans la config, on a donc spécifié une classe CustomRoleProvider, qui va permettre de récupérer les liens entre les utilisateurs et leurs rôles. Voici un exemple d'implémentation de cette classe :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Security;

namespace WCFSecurite
{
    class CustomRoleProvider : RoleProvider
    {
        public override String ApplicationName { get; set; }

        public override String[] GetRolesForUser(String username)
        {
            return new String[] { XXXXXX.getRoles(username) };
        }

        public override bool IsUserInRole(String username, String roleName)
        {
            return GetRolesForUser(username).Contains(roleName);
        }

        public override void AddUsersToRoles(String[] usernames, String[]
        roleNames)
        {
            throw new Exception("Unable to perform this action");
        }

        public override void CreateRole(String roleName)
        {
            throw new Exception("Unable to perform this action");
        }

        public override bool DeleteRole(String roleName, bool throwOnPopulatedRole)
        {
            throw new Exception("Unable to perform this action");
        }

        public override String[] FindUsersInRole(String roleName, String
        usernameToMatch)
        {
            throw new Exception("Unable to perform this action");
        }

        public override String[] GetAllRoles()
        {
            throw new Exception("Unable to perform this action");
        }

        public override String[] GetUsersInRole(String roleName)
        {
            throw new Exception("Unable to perform this action");
        }

        public override void RemoveUsersFromRoles(String[] usernames, String[]
        roleNames)
        {
            throw new Exception("Unable to perform this action");
        }

        public override bool RoleExists(String roleName)
        {
            throw new Exception("Unable to perform this action");
        }
    }
}
```

Ici, nous n'avons pas implémenté toutes les méthodes de la classe abstraite RoleProvider mais cela suffit pour nos tests. La classe XXXXXX peut par exemple définir un accès à une base de données où seraient stockés les rôles pour chaque utilisateur.

3. Création d'un client Test

On crée un client Test en mode console dans lequel on ajoute une référence vers notre service comme expliqué dans l'énoncé complémentaire 2. On ajoute ensuite le code suivant pour faire un appel aux 2 méthodes de notre service :

```
// instanciation de la référence vers le service
WCFSecuriteServiceReference.WCFServiceClient serviceReference = new
WCFSecuriteServiceReference.WCFServiceClient();

// ajout des credentials sur notre référence
serviceReference.ClientCredentials.UserName.UserName = "broco";
serviceReference.ClientCredentials.UserName.Password = "pwd";

// si le username/password est correct et que l'utilisateur broco appartient au
// groupe user
Console.WriteLine(serviceReference.getInfoPublique());
// si l'utilisateur broco appartient au groupe VIP
Console.WriteLine(serviceReference.getInfoLouche());
// sinon exception de sécurité!!!
```