

Sécuriser un Webservice en Asp.net grâce aux cookies

par Ludovic Lefort ([Site web](#)) ([Blog](#))

Date de publication : 31/08/2007

Dernière mise à jour : 31/08/2007

Description d'une méthode de sécurisation de web service utilisant les cookies.

- 0 - Introduction
- 1 - Déroulement global des opérations
- 2 - Web service de base (non sécurisé)
- 3 - Création d'une classe UserToken
- 4 - Création de la classe SecurityManager
- 5 - Identification de l'utilisateur
- 6 - Ecrire une méthode sécurisée
- 7 - Coté client
- 8 - Conclusion
- 9 - Sources de l'exemple

0 - Introduction

Dans le cas où un web service doit être publié sur le web, il paraît indispensable de le sécuriser si il est lié à des informations confidentielles.

L'utilisation de cookies permet de rendre l'application compatible avec des clients Windows Form et Web.

1 - Déroulement global des opérations

- 1 Identification de l'utilisateur (username, password).
- 2 Un token contenant des informations sur l'utilisateur est stocké dans la variable de session et un cookie est envoyé au client.
- 3 Le client envoie ce cookie lors des appels aux web services.

2 - Web service de base (non sécurisé)

Pour notre exemple nous allons partir d'un web service basique non sécurisé avec une seule méthode qui reçoit deux entiers et retourne la somme. Rien de bien méchant :)

```
[WebMethod]
public int Sum(int x, int y)
{
    return x + y;
}
```

3 - Création d'une classe UserToken

Cette classe nous permettra de stocker toutes les informations nécessaire à la vérification de la validité de l'appel au web service.

Dans cet exemple nous allons vérifier une seule chose :

- L'adresse IP : elle ne peut pas changer entre deux appels

Voici le code de la classe **UserToken**

```
public class UserToken
{
    private string _ip;

    public String Ip
    {
        get
        {
            return _ip;
        }
    }

    public UserToken(string ip)
    {
        this._ip = ip;
    }
}
```

4 - Création de la classe SecurityManager

La classe **SecurityManager** contient deux méthodes statiques :

- createUserToken
- checkUserToken

La méthode CreateUserToken instancie un objet **UserToken** en se basant sur l'ip du client et le retourne :

```
internal static UserToken createUserToken()
{
    UserToken utoken = new UserToken(HttpContext.Current.Request.UserHostAddress);
    return utoken;
}
```

Voici le code de la méthode qui permet de vérifier la validité de l'appel au web service :

```
internal static void checkUserToken(UserToken uToken)
{
    if (uToken != null)
    {
        //Est ce que l'ip a changée ?
        if (HttpContext.Current.Request.UserHostAddress != uToken.Ip)
        {
            throw new Exception("the ip has changed");
        }
    }
    else
    {
        throw new Exception("Not authorized");
    }
}
```

5 - Identification de l'utilisateur

Avant que l'utilisateur puisse faire appel à une méthode d'un web service, il doit être identifié.

Nous allons donc créer un web service d'identification :

```
[WebMethod(true)]
public bool Identify(string username,string password)
{
    //Vérification du user name et password
    bool accepted = true;// ... méthode permettant l'authentification

    if (accepted)
    {
        UserToken utoken = SecurityManager.createUserToken();
        this.Session.Add("UserToken", utoken);
    }
    return accepted;
}
```

Dans cet exemple l'identification est toujours acceptée.

Vous avez peut-être remarqué le paramètre 'true' dans l'attribut [WebMethod]

Il s'agit de la propriété **EnableSession** de l'attribut WebMethod qui permet d'accéder à la variable de session dans notre application.

Si la vérification du username et password est acceptée nous faisons appel à la méthode **SecurityManager.createUserToken()**

A partir de maintenant l'utilisateur peut faire appel aux web services.

6 - Ecrire une méthode sécurisée

Voici la version modifiée de la méthode Sum() de la première section :

```
[WebMethod(true)]
public int Sum(int x, int y)
{
    SecurityManager.checkUserToken(this.Session["UserToken"] as UserToken);
    return x + y;
}
```

Deux modifications ont été apportées : l'ajout de l'attribut **EnableSession** pour activer la variable de session et l'appel à la méthode **SecurityManager.checkUserToken()** en passant le token se trouvant dans la session pour sécuriser le reste des opérations.

7 - Coté client

Reste maintenant à décrire l'appel du client.

Dans un premier temps, il faut faire appel à un premier web service, celui qui va nous permettre l'identification.

Nous allons nommer la classe proxy **Iden**

Pour que la session soit conservée même dans le cas d'un appel depuis une application non-web, il faut la stockée dans un cookie. Pour cela nous allons utiliser un objet de type CookieContainer :

```
System.Net.CookieContainer cc;
```

Créons maintenant notre classe proxy pour le web service **Iden**

```
Iden.Identification i = new ConsoleApplication1.Iden.Identification();
```

et attachons lui le CookieContainer :

```
i.CookieContainer = cc;
```

reste maintenant à s'identifier :

```
i.Identify("username", "password");
```

Si l'identification est acceptée l'objet **cc** contiendra un cookie permettant de lier les prochains appels à la session utilisateur du serveur web.

Appelons maintenant la méthode Sum de notre web service :

```
Serv.Service1 s = new ConsoleApplication1.Serv.Service1();  
s.CookieContainer = cc;  
int x = s.Sum(2, 3);
```

Nous avons créer un classe proxy pour le web service, ensuite nous lui avons attaché le cookie container pour finalement faire appel à sa méthode Sum.

8 - Conclusion

Cette méthode de sécurisation de web service n'est évidemment pas la seule existante.

Elle a l'avantage d'offrir une solution relativement sûre avec très peu d'implémentation supplémentaire pour sécuriser un web service existant.

De plus elle permet d'être appelée depuis une application web ou windows.

9 - Sources de l'exemple

Téléchargez les **sources** de cet exemple

Merci à **fabszn** pour la relecture de cet article.

