



# Sécuriser ses applications Web et réseaux

(Avec code source à télécharger)

*Mis à jour le 10/05/2005*  
Par Xavier Vanneste

## Sécuriser ses applications Web et réseaux

### Introduction

La sécurité est à la mode en ce moment, tout doit être sécurisé, comme si une conscience collective s'était rendu compte à quel point la sécurité avait une importance fondamentale dans le développement d'une application.

Le problème c'est qu'à l'heure actuelle, peu d'entreprise se donne les moyens de former les développeurs à la sécurité. Beaucoup pense que la sécurité est une affaire d'infrastructure. "La sécurité ce n'est pas un problème j'ai un firewall", voici typiquement le genre de réponse que j'ai lorsque je parle sécurité avec des responsables informatiques.

Mais voilà si l'infrastructure est infranchissable mais que le développement est remplis de faille de sécurité alors c'est un peu comme si vous habitiez une forteresse hyper sécurisé en surface mais avec plein de sous terrain a l'air libre et ouvert à tout le monde.

Au cours de cet article je vais vous présenter un exemple (les sources sont disponibles sur le site) remplis d'erreur. Cet exemple est le stéréotype de ce qu'il ne faut pas faire. Peut être que certaines erreurs vous sembleront grossière et limite vous penserez que je me suis moqué de vous. Mais sachez que pour toute erreur que vous trouverez simple d'autre ne la verront même pas.

Inversement les erreurs que vous ne verrez pas, d'autre les trouveront ridicule.

Le but n'est pas non plus de faire une morale ou de se prendre pour le messie de la sécurité. Je ne suis pas infallible et j'en découvre tous les jours. Le but est juste de vous montrer que chaque ligne de code tapée peut devenir une arme destructrice et se retourner contre nous.

### Les erreurs à ne pas commettre

#### Ce qu'on se dit:

-Ca m'arrivera jamais?

**Ben si justement ça arrive à tout le monde pas aux autres plus qu'à vous**

-Qui fait ça?

**Les hackers, les scripts kiddies, les pirates informatiques bref y a plein de monde qui le fait. Ce n'est pas parce qu'on ne le fait pas que personne ne le fait.**

-Pour qu'ils y arrivent faudrait que ça soient des génies?

**Ce n'est pas parce que quelque chose nous semble impossible que ça l'est. Une personne qui a une bonne expérience en sécurité prouve facilement que des fois c'est très simple**

-Mon réseaux me protégera

**Une forteresse aux portes de verre voilà à quoi vous vous exposez. Un réseau qui est pénétré par une faille**

dans votre développement.

-On n'a pas les outils qui nous permettent de nous protéger

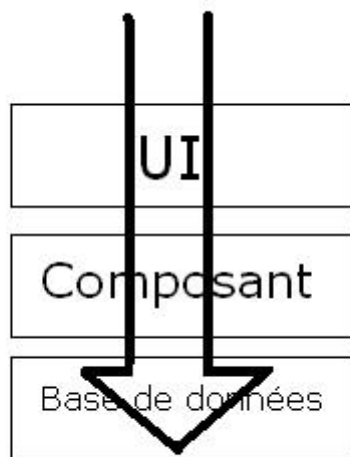
**Un mauvais ouvrier n'a que de mauvais outils comme me disait mon père. Le meilleur outil pour la sécurité c'est votre tête. Des outils comme NUNIT doivent être paramétrés pour fonctionner et avant de les paramétrer il faut savoir quoi mettre.**

### La façon dont on sécurise:

Beaucoup de personnes sécurisent de haut en bas, c'est à dire qu'elles sécurisent d'abord l'UI puis si elles arrivent encore à passer elles sécurisent les composants et enfin la base de données.

Le problème de cette façon de sécuriser c'est qu'on ne sécurise un niveau inférieur que si on arrive à passer un niveau supérieur. Mais la question qui se pose est: "si vous n'arrivez pas à passer l'UI est ce que ça veut dire que personne ne sera la passer?"

Voici le schéma utilisé:



Il est clair maintenant que la façon de sécuriser doit être inversée, sécuriser d'abord la base de données puis les composants et enfin l'interface utilisateur. Ainsi vous aurez une bonne vision de ce qui est faisable sur votre application quelque soit le niveau ou l'attaquant arrive.

### Les grands principes

#### SD3+C

- **Sécurisé par design** La sécurité doit faire partie dès la conception, cela doit être vu comme une fonctionnalité de l'application pas une valeur ajoutée. La conception de la sécurité doit se retrouver à tous les niveaux de votre application de la base de données à l'interface utilisateur.  
Le fait d'identifier les attaques possibles dès la conception permet de préparer les tests qu'il faudra effectuer sur l'application ainsi que de concevoir dès le départ les parades possibles à chaque attaque
- **Sécurisé par défaut** La plupart des personnes, lorsqu'elles installent un logiciel font le fameux SUIVANT/SUIVANT.../TERMINER, et ce sans se poser la moindre question. Elles font confiance au logiciel d'installation, et ne lisent même pas la notice livrée avec le logiciel. Résultat, il y a quelques années Code Rouge a fait des ravages en se propageant à grande vitesse. Code rouge était un vers qui utilise une faille de IIS pour faire un buffer overflow (voir annexe). A cette époque IIS, le serveur web de Windows, était proposé par défaut dans l'installation. Comme tout le monde prend les paramètres par défaut tous ceux qui avaient IIS se trouvèrent infectés par Code Rouge. Il est donc important de réduire la surface d'attaque et de proposer le minimum de fonctionnalités nécessaires à l'application pour fonctionner, afin d'éviter d'installer des fonctionnalités qui pourraient cacher des failles potentielles.  
Il faut aussi faire attention à réduire les risques sur les fonctionnalités installées. Prenons un autre exemple SQLWorm est un vers qui pénètre les serveurs SQL sur le port

d'administration. Dans les paramètres par défaut, le port d'administration est fixe. Donc sqlworn scrute les adresses IP sur le port d'administration qu'il connaît. Comme tout le monde installe par défaut tous ceux qui avaient laissés le port d'administration par défaut se sont retrouvés infesté. Ceux qui avaient eu la bonne idée de changer ce port n'ont pas été touchés par le vers. Mais l'installation aurait du créer un port d'administration de manière aléatoire comme ça SQLWorn aurait non seulement du scruter les adresses IP mais aussi tous les ports qui existent soit plus de 65000. Sa propagation aurait été moins rapide.

La sécurisation par défaut nécessite une bonne étude de l'application et une bonne documentation pour l'utilisateur. Qui la lira comme d'habitude une fois l'application installée ;- ) mais il faut lui laisser la possibilité de pouvoir revenir en arrière et enlevé facilement des fonctionnalités dont il n'a pas besoin

- **Sécurisé par déploiement** Voila notre application déployée chez les clients il faut maintenant pouvoir la maintenir, c'est à dire avoir la possibilité de réparer facilement les bugs et les failles de sécurité à l'aide de patch. Il faut aussi que cette procédure ne soit pas trop compliquée pour l'utilisateur. Avec Click Once sous visual studio 2005 les choses seront grandement facilitées
- **Sécurisé par communication** A quoi ça sert d'avoir trouvé une faille de l'avoir corrigé et d'avoir mis un patch en ligne si personne ne le sait? La sécurité par la communication passe par là. Il faut faire suivre l'information et la propager.

### Sécurisé en profondeur

Qui peut dire ma couche interface utilisateur est assez sécurisé, je n'ai pas besoin de sécurisé les composants et la base de données? Je pense que personne ne peut dire cela. Le but de sécurisé en profondeur n'est pas de rendre une application infaillible (aucune ne l'est on en a la preuve tous les jours) mais de la rendre plus dur à attaquer. La cible de la plupart des attaques c'est les données.

Qu'elles soient dans des fichiers ou sur des bases de données la plupart du temps le but et d'accéder a des données. Il faut donc les considérer comme un coffre fort dans une banque.

Quand vous entrez dans une banque, vous passez d'abord un sas pour voir si vous n'êtes pas un cambrioleur. Ensuite vous avez un guichet, et enfin le coffre. Dans une banque toutes les parties du sas au coffre fort sont sécurisées.

Dans votre application ça doit être pareil. La sécurité de la base de données, des composants qui y accèdent, de l'interface utilisateur, tout doit être vérifié. Quand on parle de sécurité on parle de tous les aspects de la sécurité: autorisation, authentification et audit.

Chaque appel à un autre serveur doit faire l'objet d'un firewall, chaque composant doit vérifier l'identité de l'appelant d'une de ses méthodes, chaque source de données doit vérifier les droits de celui qui fait les requêtes.

On pourrait croire que la sécurité en profondeur est une perte de temps ou que ça ne sers à rien voir que ça pénéliserait l'administration, les performances et la maintenance de l'application.

En fait il ne faut partir du principe que la sécurité c'est tout coupé. La sécurité c'est vérifier que la personne avec qui l'application dialogue est la bonne personne. Dans ce cas l'administration et la maintenance sera facilité car on aura moins d'attaque qui arriveront au but.

### Identifier les attaques (STRIDE)

L'identification des attaques ainsi que leur impacte sur l'application doit très tôt faire partie du cycle de développement de toute application.

Ci-dessous ce tableau vous présente STRIDE, c'est les différentes catégories d'attaque possible.

<b>S</b>	<b>Spoofing</b>	Se faire passer pour un autre utilisateur. Exemple passer la page de login avec des autres informations utilisateurs
<b>T</b>	<b>Tampering with data</b>	Modification de données
<b>R</b>	<b>Repudiation</b>	Faire une action qui ne nous est pas permise normalement. On est bien authentifié mais une partie de l'application nous permet de faire quelque chose qu'on n'est pas senser pouvoir faire
<b>I</b>	<b>Information Disclosure</b>	Les informations sensibles du site sont visibles.

		Par exemple en écoutant la trame réseaux et en voyant en clair les informations circuler
<b>D</b>	<b>Denial of services</b>	Empêcher les utilisateurs d'accéder à l'application
<b>E</b>	<b>Elevation of privilege</b>	Pouvoir utiliser un compte ayant beaucoup de privilèges dans l'application alors qu'on ne peut normalement pas le faire. Exemple utiliser le compte administrateur sur un site où on a même pas les droits invités normalement.

Prenons un exemple une injection de code sql sur un système pas du tout sécurisé:

<b>S</b>	L'identité peut être subtilisée à la page de login
<b>T</b>	Les données dans la base peuvent être modifiées
<b>R</b>	On a la possibilité d'accéder et de modifier des données qui ne nous sont pas permises en fonctionnement normal
<b>I</b>	Récupération d'information sensible
<b>D</b>	Destruction de la base qui entraîne l'arrêt de l'application
<b>E</b>	On se logue en tant qu'administrateur

L'impact sur le site est conséquent.

Prenons l'exemple du Cross Site Scripting

<b>I</b>	On peut lire les cookies du site ou en écrire pour récupérer des informations sur l'utilisateur
<b>D</b>	On redirige les clients sur un autre site ou on rend la page vulnérable inaccessible

L'impact sur le site est moindre

## Présentation du site à sécuriser

Tout au long de l'article je vais utiliser un site pas du tout sécurisé puis le sécuriser petit à petit en vous montrant les exemples à faire et ce à ne pas faire.

Les pages login.aspx et inscrire.aspx font partie de la partie non sécurisée (en effet pour s'inscrire il ne faut pas encore être inscrit et pour se connecter il faut pas encore être connecté). Les pages default file et guestbook ainsi que informations font partie de la partie sécurisée remarquez que la page information est une page HTML et la page admin.aspx fait partie de la zone sécurisée administrateur.

Sur le site les rudiments sont posés (voir le code). C'est à dire qu'en apparence tout fonctionne mais en apparence seulement.

Le but du site est assez bateau et pour l'article je n'ai pas fait un site exceptionnel pour la bonne raison que le but est de montrer la sécurité et pas de faire des exercices de styles dans le code.

La personne se connecte sur le site, arrivera sur la page d'accueil, elle aura la possibilité d'aller sur la page livre d'or, ou sur la page de gestion de ses fichiers.

Sur la page du livre d'or elle peut laisser un message pour les utilisateurs du site.

Sur la page de gestion de fichiers un répertoire est créé pour l'utilisateur s'il n'existe pas, puis l'utilisateur peut voir ses fichiers dans une combobox, ou taper directement le nom du fichier dans une textbox. Si le fichier qu'il a tapé dans la textbox n'existe pas il est créé. Dans un textarea le contenu du fichier est affiché et l'utilisateur peut modifier et mettre à jour ce contenu.

Sur la page d'administration on n'affiche que les utilisateurs. Le but de cette page est juste de montrer comment sécuriser une petite partie du site.

La page d'inscription comporte un formulaire qui permet aux personnes de s'inscrire sur le site. En plus des pages web, un web service est un composant .net Remoting hoster par IIS sont présent.

Le web service retourne un objet CUser correspondant au login passé en paramètre, l'objet remoting renvoie les informations utilisateurs, les informations du serveur et permet la mise a jours du mot de passe d'un utilisateur par rapport a son login. Comme vous pouvez le voir il n'y a rien de transcendant dans ce site.

## Les failles

Pour montrer les failles je commencerais par expliquer ce que l'utilisateur est sensé faire puis je montrerais ce qu'un hacker peut en faire, je ne rentrerai pas dans le détail des attaques on le verra un peu plus loin:

### La page login

C'est LA page par excellence qu'il faut sécuriser. C'est elle qui identifie l'utilisateur et c'est avec cette authentification qu'on fera les autorisations (ne pas confondre autorisation et authentification l'authentification ça répond à la question qui est la, l'autorisation répond à la question qu'est ce que je peux faire).

La page login se base sur la base de données pour rechercher les informations sur l'utilisateur. Le bout de code suivant montre comment les informations sont récupérées.

```
private void btnOk_Click(object sender, System.EventArgs e)
{
    _objSqlCnx=new
SqlConnection(ConfigurationSettings.AppSettings.Get("sqlCNXString"));
    _objSqlCnx.Open();
    _objSqlCommand=new SqlCommand("select count(*) from utilisateurs where Login='"
+ txtLogin.Text + "' and mdp='" + txtPassword.Text + "'",_objSqlCnx);
    int _count=(int) _objSqlCommand.ExecuteScalar();
    if(_count>0)
    {
        System.Web.Security.FormsAuthentication.RedirectFromLoginPage(txtLogin.Text
,false);
        CUser objUser=(CUser) Session["objUser"];
        objUser.Login=txtLogin.Text;
    }
}
```

#### Quel est le problème dans ce code?

Regardez bien la ligne suivante:

```
_objSqlCommand=new SqlCommand("select count(*) from utilisateurs where Login='" +
txtLogin.Text + "' and mdp='" + txtPassword.Text + "'", objSqlCnx);
```

On passe ce que l'utilisateur a tapé directement dans la base de données. Le résultat est qu'il peut taper du code SQL celui ci passera, c'est ce qu'on appel de l'injection de code SQL, on y reviendra par la suite.

### La page guestbook

La page guestbook est un livre d'or qui permet a une personne d'entré du texte qui sera réaffiché après. Ce texte peut être du code HTML étant donné que pour des raisons de "facilités" d'utilisation le développeur a mis dans l'en-tête de la page ceci:

```
<%@ Page language="c#" Codebehind="GuestBook.aspx.cs" ValidateRequest="false"
AutoEventWireup="false" Inherits="NonSecure.GuestBook" %>
```

La, il faut dire que le développeur n'a pas beaucoup réfléchi. Il a essayé d'entrer du texte, celui ci a planté, il a vu qu'il devait mettre ValidateRequest à false pour que les < passent il a donc décidé de le mettre.

**Quel est le problème?** Regardons le code pour insérer les données en base de données:

```
private void btnAdd_Click(object sender, System.EventArgs e)
{
    _objSqlCnx=new
SqlConnection(ConfigurationSettings.AppSettings.Get("sqlCNXString"));
    _objSqlCnx.Open();
    _objSqlCommand=new SqlCommand("insert into guestbook(IDUser,contenu) values ("
```

```
_objSqlCommand.ExecuteNonQuery();
objdadata.Fill(monDS1);
DataList1.DataBind();
}
```

Le développeur ne fait qu'entrer ce qui a été tapé dans la textarea dans la base de données. Imaginé que ce qui a été tapé dans la textarea soit ceci:

```
<script language="javascript">
document.location.href="http://www.sex.com";
</script>
```

Dans ce cas que ce passera t'il lorsque la page se réaffichera?

Le code javascript ainsi insérer dans le guestbook sera exécuté et la page du site pornographique sera affichée. Imaginez qu'il y ait lecture d'un cookie ou envoi d'information de la personne connectée sur un autre serveur et on tombe dans la violation de la vie privée. C'est ce qu'on appelle du cross site scripting on verra comment se défendre et comment cela fonctionne par la suite.

## La page file

La page file semble ne pas contenir de défaut. Elle ne fait que lire des fichiers qui se trouvent dans un répertoire, elle peut aussi récupérer des fichiers qui sont uploader par l'utilisateur pour que celui ci les mettent dans son répertoire à lui. Le premier problème vient du fait que l'utilisateur peut taper le nom de son fichier. Voyons le code qui est mis en place pour la lecture du fichier:

```
private void btnOK_Click(object sender, System.EventArgs e)
{
    if(!File.Exists(_objUser.Chemin + "\\\" + txtOther.Text))
    {
        StreamWriter objWriter=File.CreateText(_objUser.Chemin + "\\\" +
txtOther.Text);
        objWriter.Flush();
        objWriter.Close();
        ddlFile.DataSource=_objUser.Fichiers;
        ddlFile.DataBind();
        ddlFile.Items.Insert(0,"Nom du fichier");
        ddlFile.Items.FindByText(txtOther.Text).Selected=true;
    }
    else
    {
        StreamReader objStream= new StreamReader(_objUser.Chemin + "\\\" +
txtOther.Text);
        txtContent.Text =objStream.ReadToEnd();
        objStream.Close();
    }
}
```

### Où est le problème?

Le problème est qu'on passe directement le nom du fichier au system de fichier. Par défaut les fichiers se trouve dans /file/nomUser par rapport à la racine du site.

Imaginez que quelqu'un tape ../..\Web.config dans ce cas le system de fichier irait dans \file\username\..\web.config. On afficherait donc directement le code du fichier Web.Config dans la textbox.

C'est ce qu'on appelle une erreur de canonisation de nom de fichier.

Le second problème vient du fait qu'on peut uploader n'importe quoi, c'est à dire que quelqu'un qui upload avec 30 machines différentes des fichiers de 100Mo bloquera la bande passante du serveur et créera un DOS (Denial Of Service)

## La page d'information

La page d'informations Information.html semble être en sécurité. Elle est en HTML donc le serveur ne craint rien, elle est destinées aux utilisateurs du site, elle se trouve donc dans le répertoire sécurisé sous la protection de l'authentification du fichier web.config

### Où est le problème?

Le problèmes est que la page a l'extension HTML donc IIS ne va pas le soumettre au moteur ASP.NET donc si on tape l'adresse de la page HTML on tombera sur la page HTML sans avoir à passer par la page d'authentification

## Les erreurs sournoises

### Le fichier global.asax

Regardons le fichier global.asax, dans le session start le développeur créé en variable de session

un objet de type CUSER afin de stocker les informations des utilisateurs authentifiés sur le reste du site:

```
protected void Session_Start(Object sender, EventArgs e)
{
    CUser objUser= new CUser();
    Session["objUser"]=objUser;
}
```

#### Où est le problème?

Le problème est qu'on alloue de la mémoire pour un utilisateur qui n'est pas authentifié, si un hacker envoie 3000 Sessions différentes on allouera 3000 Objets pour rien

#### La page file

Dans la page file.aspx on affiche les fichiers texte postés par un utilisateur. Seul l'utilisateur qui a posté le fichier est censé voir ce fichier. Ces fichiers se trouvent dans les sous répertoires utilisateurs du répertoire file.

#### Où est le problème?

Le problème est que le répertoire file est accessible depuis le Web. Alors que ce passe t'il si quelqu'un tape le nom d'un des fichiers dans l'URL de la page. Le fichier s'ouvre dans Internet Explorer, tout simplement. Pour résoudre ce problème on va mettre en place un handler sur l'extension TXT pour empêcher quelqu'un de mal intentionné de voir les fichiers TXT des utilisateurs du site. **Résolution des problèmes et sécurisation du site** Afin de résoudre les différents problèmes du site j'en ai fait une copie (le code est dans le projet secure). Pour la sécurisation du site on partira des données puis on sécurisera chaque partie.

#### La page Login

On commence par sécuriser la page login. Où était le problème? Le problème venait du passage des saisies de l'utilisateur dans les requêtes SQL sans vérification de ce qui est passé. L'utilisateur peut injecter du code SQL simple (requête select classique), mais que ce passe t il s'il envoie une requête DDL comme un alter voir un drop table? Que ce passe t il si la requête DML qu'il envoie est un update ou un delete? Posons nous maintenant les bonnes questions:

#### Quel droit ai-je besoin sur la page de login pour qu'elle fonctionne?

**Réponse** : droit en lecture sur les données de la table des utilisateurs

#### Quels sont les caractères qui ne doivent pas être inclus dans la saisie utilisateur?

**Réponse** : tout caractère qui pourrait faire partie d'instruction SQL

#### Quels sont les caractères que l'utilisateur peut saisir alors?

**Réponse** : tous caractères alphanumériques mais sans signe pour la page de login

#### Comment réagir si une injection de code SQL arrive quand même à entrer?

**Réponse** : il ne faut pas tout baser sur la requête SQL alors car on ne peut pas lui faire confiance.

Réparons la page Login en commençant par la base de données (schéma de sécurisation ascendant et non descendant).

Premièrement nous allons regarder ce qu'il y a dans le fichier Web.config concernant la chaîne de connexion sur le serveur.

```
<appSettings>
  <add key="sqlCNXString" value="Data Source=SPARTNER01XP;initial
catalog=NonSecureDB;user id=sa; pwd="/>
</appSettings>
```

Pas terrible comme chaîne de connexion, on accède au serveur SQL comme superadministrateur et on peut faire ce que bon nous semble.

Créons donc un utilisateur qui a un accès en lecture seul et affectons-le à la page login.

Je crée dans SQL Server un utilisateur ReadOnlyUser qui n'a aucun rôle serveur et qui est associé au rôle public, db\_datareader, db\_denydatawriter de ma base de données securedb de plus cet utilisateur à un mot de passe.

Ajoutons la chaîne de connexion au web.config et affectons-la à la page login:

```
<appSettings>
  <add key="sqlCNXString" value="Data Source=SPARTNER01XP;initial
catalog=SecureDB;user id=sa; pwd="/>
  <add key="sqlCNXStringReadOnly" value="Data Source=SPARTNER01XP;initial
catalog=SecureDB;user id=ReadOnlyUser; pwd=LectureSeule"/>
```

```

</appSettings>
_objSqlCnx=new
SqlConnection(ConfigurationSettings.AppSettings.Get("sqlCNXStringReadOnly"));

```

Bien maintenant passons à un cran au dessus et sécurisons l'accès aux données.

On doit faire en sorte qu'on ne puisse pas passer du code SQL au requête. Une des solutions est la procédure stockée, sinon on a les requêtes paramétrées. Afin de contrer une éventuelle injection de code SQL on fera la vérification en deux temps, on testera la présence du login dans la base mais on validera le mot de passe en ASP.Net. Voici le code.

```

private void btnOk_Click(object sender, System.EventArgs e)
{
    _objSqlCnx=new
SqlConnection(ConfigurationSettings.AppSettings.Get("sqlCNXStringReadOnly"));
    _objSqlCnx.Open();
    _objSqlCommand=new SqlCommand("select Mdp from utilisateurs where
Login=@Login", _objSqlCnx);
    _objSqlCommand.Parameters.Add("@Login", SqlDbType.VarChar);
    _objSqlCommand.Parameters["@Login"].Value=txtLogin.Text;
    string _Mdp=(string) _objSqlCommand.ExecuteScalar();
    if (_Mdp==txtPassword.Text)
    {
        System.Web.Security.FormsAuthentication.RedirectFromLoginPage(txtLogin.Text
, false);
        CUser objUser=(CUser) Session["objUser"];
        objUser.Login=txtLogin.Text;
    }
}

```

Dans ce code on utilise les requêtes paramétrées et afin de contrer toute injection de code SQL possible en Select (les autres sont impossibles du à la sécurité mise sur la base de données) on créer une vérification en deux temps une en sql permettant de renvoyer le login et une en asp.net pour verifier le mot de passe. Celle en ASP.Net ne sera pas, par définition, sensible à l'injection de code SQL.

Continuons notre sécurisation de la page login en s'attaquant maintenant au niveau le plus haut (sécurisation ascendante toujours) c'est à dire l'interface utilisateur.

On va mettre un regular expression validateur sur les champs login et mot de passe en obligeant l'utilisateur à ne rentrer que des lettre ou des chiffres.

```

<asp:RegularExpressionValidator id="RegularExpressionValidator2" runat="server"
    ErrorMessage="Chiffre ou lettre uniquement pour le mot de passe"
Display="Dynamic" ControlToValidate="txtPassword" ValidationExpression="\w+"/>
<asp:RegularExpressionValidator id="RegularExpressionValidator1" runat="server"
    ErrorMessage="Chiffre ou lettre uniquement pour le login"
    Display="Dynamic" ControlToValidate="txtLogin" ValidationExpression="\w+"/>

```

Voilà qui est fait. Mais le souci est que pour l'instant la vérification se fait côté client. Côté serveur on a pas de code qui permet de faire la vérification. Imaginez que l'utilisateur désactive les javascripts les données seront envoyé au serveur. Les validateurs font une validation coté client et empêche la page de se poster. Mais côté serveur les validateurs n'empêchent pas l'exécution du code. Par contre ils modifient la propriété de page IsValid. Voici le code définitif

```

private void btnOk_Click(object sender, System.EventArgs e)
{
    if (Page.IsValid)
    {
        _objSqlCnx=new
SqlConnection(ConfigurationSettings.AppSettings.Get("sqlCNXStringReadOnly"));
        _objSqlCnx.Open();
        _objSqlCommand=new SqlCommand("select Mdp from utilisateurs where
Login=@Login", _objSqlCnx);
        _objSqlCommand.Parameters.Add("@Login", SqlDbType.VarChar);
        _objSqlCommand.Parameters["@Login"].Value=txtLogin.Text;
        string _Mdp=(string) _objSqlCommand.ExecuteScalar();
        if (_Mdp==txtPassword.Text)
        {
            System.Web.Security.FormsAuthentication.RedirectFromLoginPage(txtLogin.Text
, false);
        }
    }
}

```

```
        objUser.Login=txtLogin.Text;
    }
}
}
```

## La page guestbook

La page guestbook est sensible au Cross Site scripting, c'est à dire que si quelqu'un met du HTML dans la page, le html sera stockée en base et sera renvoyé à la page guestbook sans modification. Il est donc possible de mettre n'importe quoi même du script javascript.

Comment résoudre le problème puisqu'on souhaite quand même laisser la possibilité à l'utilisateur de mettre du code HTML dans la page?

Il faut donc faire un filtre sur le texte récupéré de la textbox sans oublier de sécuriser la page contre l'injection de code SQL.

Posons-nous les bonnes questions:

### Quels droits ai je besoin sur ma base de données?

**Réponse:** dans la base on fait de la lecture à un endroit et de l'écriture à un autre endroit. Il faut donc voir si on a une connexion a la base ou plusieurs. Si on en a plusieurs on en fera une en écriture seul et une en lecture seule

### Que peut saisir l'utilisateur dans la textbox?

**Réponse:** le problème dans cette page c'est qu'il peut tout saisir, signe, chiffre, lettre, il devient alors difficile de savoir ce qu'il faudra lui autoriser. On va donc se baser sur des requêtes paramétrées pour pouvoir limiter les chances d'injections de code SQL.

### Quelle balise HTML est ce que j'accepte?

**Réponse:** Il faudra supprimer toute balise de script, de lien, d'image, tout événements javascript potentiel, formulaire, bref il ne faudra garder que les balises de mise en forme a savoir bold, souligné, italique, tableau, et supprimer toutes les autres. En partant du principe, j'accepte rien au départ puis autorise petit a petit et non j'accepte tout puis supprime car si on oublie quelque chose c'est plus ennuyeux dans le second cas.

Sécurisons la page guestbook en commençant par créer un utilisateur en écriture seul. Cet utilisateur n'aura pas de rôle serveur et aura comme rôle de base de données sur SecureDB public, db\_datawriter, db\_denydatareader. Il sera donc en écriture seul. J'ajoute la chaîne de connexion dans le Web.Config.

```
<appSettings>
  <add key="sqlCNXString" value="Data Source=SPARTNER01XP;initial
catalog=SecureDB;user id=sa; pwd="/>
  <add key="sqlCNXStringReadOnly" value="Data Source=SPARTNER01XP;initial
catalog=SecureDB;user id=ReadOnlyUser; pwd=LectureSeule"/>
  <add key="sqlCNXStringWriteOnly" value="Data Source=SPARTNER01XP;initial
catalog=SecureDB;user id=WriteOnlyUser; pwd=EcritureSeule"/>
</appSettings>
```

Je mets l'utilisateur en lecture seul pour la création de mon datalist et je met l'utilisateur en écriture seul pour la création des données en base.

```
this.objcndata.ConnectionString =
((string) (configuration.AppSettings.GetValue("sqlCNXStringReadOnly",
typeof(string))));
_objSqlCnx=new
SqlConnection(ConfigurationSettings.AppSettings.Get("sqlCNXStringWriteOnly"));
```

Bien maintenant faisons une requête paramétrée pour le code qui est inséré en base de données, remarquez que les paramètres sont typé ce qui ajoute à la sécurisation de la requête. En effet on ne pourra pas envoyer du texte sur un paramètre de type int, ce qui veut dire pas de code SQL.

```
private void btnAdd_Click(object sender, System.EventArgs e)
{
    _objSqlCnx=new
SqlConnection(ConfigurationSettings.AppSettings.Get("sqlCNXStringWriteOnly"));
    _objSqlCnx.Open();
    _objSqlCommand=new SqlCommand("insert into guestbook(IDUser,contenu) values
(@IDUser,@Contenu)",_objSqlCnx);
    _objSqlCommand.Parameters.Add("@IDUser",SqlDbType.Int);
    _objSqlCommand.Parameters.Add("@Contenu",SqlDbType.VarChar);
```

```

_objSqlCommand.Parameters["@Contenu"].Value=txtGuestContent.Text;
_objSqlCommand.ExecuteNonQuery();
objdadata.Fill(monDS1);
DataList1.DataBind();
}

```

Bien maintenant attaquons nous aux filtres sur les balises HTML. Le problème sur le filtre c'est qu'on ne peut pas envoyer un filtre sur <script par exemple car la personne pourrait rentrer <script en mettant un espace résultat le filtre ne fonctionnerait plus. Voici la solution que je propose mais il doit y en avoir d'autre:

```

private void btnAdd_Click(object sender, System.EventArgs e)
{
    _objSqlCnx=new
SqlConnection(ConfigurationSettings.AppSettings.Get("sqlCNXStringWriteOnly"));
_objSqlCnx.Open();
string[]
_tblStrHtml=txtGuestContent.Text.Replace("<","&lt;").Replace(">","&gt;").Split('#');
ArrayList objArrLst=new ArrayList(_tblStrHtml);
foreach(string str in _tblStrHtml)
{
    string strTemp=str.ToLower();
    if(strTemp.IndexOf("<")>=0)

    if(!(System.Text.RegularExpressions.Regex.Match(strTemp,"(?:</?(?:i|b|p|br|em|pre|table|td|tr){1}>").Success))
        objArrLst.Remove(str);
}
_tblStrHtml=(string[])objArrLst.ToArray(typeof(string));
_objSqlCommand=new SqlCommand("insert into guestbook(IDUser,contenu) values
(@IDUser,@Contenu)",_objSqlCnx);
_objSqlCommand.Parameters.Add("@IDUser",SqlDbType.Int);
_objSqlCommand.Parameters.Add("@Contenu",SqlDbType.VarChar);
_objSqlCommand.Parameters["@IDUser"].Value=_objUser.IDUser;
_objSqlCommand.Parameters["@Contenu"].Value=String.Join("#",_tblStrHtml);
_objSqlCommand.ExecuteNonQuery();
objdadata.Fill(monDS1);
DataList1.DataBind();
}

```

Je crée un tableau qui est créé en séparant les balises HTML du contenu, je parcours ce tableau et à l'aide d'une expression régulière j'autorise que certains mots (on ferme tout et on ouvre petit à petit).

Je finis par recréer ma chaîne de caractères avec la méthode join.

## La page file

La page file est sensible à la canonisation de nom de fichiers en fait le problème sur cette page est qu'on peut facilement accéder à des fichiers sur le disque dur étant donnée que le code se contente de passer le nom du fichier taper au système d'exploitation.

On pourrait résoudre ce problème en décidant de rechercher les ..\ et les supprimer, mais l'utilisateur pourrait envoyer ..%5C a la place. Ce code correspond au code URL du ..\, il serait automatiquement décodé partie serveur et on ne le verrait pas passer avec notre recherche sur ..\.

De plus cette page soulève un autre problème, les informations du fichier Web.Config sont stockées de manière trop claire, il faut les masquer et les rendre moins lisible.

Posons nous les bonnes questions:

### Quel répertoire peut être accessible par la page file?

**Réponse:** Le seul répertoire qui devra être accessible par la page file sera le répertoire spécifique à l'utilisateur. Il faudra sécuriser le serveur web pour l'empêcher de remonter dans la hiérarchie des répertoires et vérifier où se trouve le fichier qu'on affiche.

### Où stocker les données sensible pour être sûr de ne pas les exposer accidentellement au public?

**Réponse:** le système de fichier traditionnel est trop critique pour stocker les chaînes de connexion de la base de données. En les cryptant il sera impossible de lire les données stockées dans le fichier Web.Config de manière simple.

### Quelle taille de fichier j'autorise en upload?

**Réponse:** notre page d'upload lit des fichiers textes on va donc limiter les tailles de fichiers textes à 2 Mo, ce qui est déjà beaucoup pour un fichier texte

### Quel type de fichier j'autorise?

**Réponse:** notre page d'upload ne prend que les fichier texte donc avec l'extension TXT.

Commençons par sécuriser le plus sensible à savoir le fichier Web.Config. Pour sécuriser le fichier Web.config on va crypter les chaînes de connexion afin de les rendre illisibles naturellement. Pour cela on va se baser sur DPAPI qui est une api du system Win32. Le code permettant d'encapsuler l'API sous .Net est inclus dans le code source du projet. Avec cette DLL je créé un client prenant en paramètre le nom du fichier Web.Config puis celui de la clé crypter.

Voici le code du client:

```
static void Main(string[] args)
{
    //
    // TODO: Add code to start application here
    //
    XmlDocument objDom=new XmlDocument();
    objDom.Load(args[0]);
    XmlNode objNode=objDom.DocumentElement.SelectSingleNode("//appSettings");
    DPAPI objApi=new DPAPI(DPAPI.Store.USE_MACHINE_STORE);
    ASCIIEncoding objEncoding=new ASCIIEncoding();
    string strEncrypter="";
    foreach(XmlNode objNodeChild in objNode.ChildNodes)
    {
        if(objNodeChild.Name=="add" &&
objNodeChild.Attributes["key"].Value==args[1])

        strEncrypter=objNodeChild.Attributes["value"].Value=Convert.ToBase64String
(objApi.Encrypt(objEncoding.GetBytes(objNodeChild.Attributes["value"].Value), null))
;
    }
    objDom.Save(args[0].Replace(".config", "Enc.config"));
    return;
}
```

Je crypte donc mes chaînes de connexions voici le fichier Web.Config maintenant.

```
<appSettings>
  <add key="sqlCNXString" value="Data Source=SPARTNER01XP;initial
catalog=SecureDB;user id=sa; pwd="/>
  <add key="sqlCNXStringReadOnly"
value="AQAAANCmnd8BFdERjHoAwE/C1~..~6FAAAID91L7d5Pf6LE7ock/Ae8V+ZQoR"/>
  <add key="sqlCNXStringWriteOnly"
value="AQAAANCmnd8BFdERjHoAwE/C1~..~4AFAAAAFsk+FSwZvb9I9DI3uJzXXwn1fQq"/>
</appSettings>
```

Bien maintenant je lis les chaînes cryptées et je les décrypte pour les passer à mes connexions bases de données dans le code, ce qui donne dans le code de la page login.

```
DPAPI objApi=new DPAPI(DPAPI.Store.USE_MACHINE_STORE);
ASCIIEncoding objEncoding=new ASCIIEncoding();
string
cnxDcrypt=objEncoding.GetString(objApi.Decrypt(Convert.FromBase64String(ConfigurationSettings.AppSettings.Get("sqlCNXStringReadOnly")), null));
objSqlConnection=new SqlConnection(cnxDcrypt);
```

Le fichier de configuration est un peu plus sécurisé. Mais le problème principale vient du fait que j'ai une faille au niveau du code de mon application.

Nous allons donc mettre en place la sécurité au niveau de mon code pour empêcher que la page file.aspx puisse lire ce qu'elle n'est pas sensée pouvoir afficher. Cette sécurité est ce qu'on appel le CAS ou Cade Access Security, on en verra une partie ici et une seconde avec la page d'administration. Commençons par la partie File.aspx. Examinons les instructions qui lisent le fichier et l'affiche dans la textarea:

```
private void btnOK_Click(object sender, EventArgs e)
{
    if(!File.Exists(_objUser.Chemin +"\\\" + txtOther.Text))
    {
        StreamWriter objWriter=File.CreateText(_objUser.Chemin +"\\\" +
txtOther.Text);
        objWriter.Flush();
    }
}
```

```

        ddlFile.DataSource=_objUser.Fichiers;
        ddlFile.DataBind();
        ddlFile.Items.Insert(0,"Nom du fichier");
        ddlFile.Items.FindByText(txtOther.Text).Selected=true;
    }
    else
    {
        StreamReader objStream= new StreamReader(_objUser.Chemin +"\\\" +
txtOther.Text);
        txtContent.Text =objStream.ReadToEnd();
        objStream.Close();
    }
}

```

Quel faut il comme permission a asp.net pour pouvoir exécuter ce code? il faut un accès en lecture écriture sur le répertoire file et tous ses sous répertoires, pour pouvoir créer les répertoire utilisateurs et pouvoir créer les fichiers. Par contre ASPNET n'a pas besoin d'accès en écriture sur la racine du site. On va donc enlever les droit d'écriture sur la racine du site et mettre les droits d'écriture sur file et ses sous répertoire.

Pour cela on va dans les propriétés du répertoire racine (clic droit propriété) onglet sécurité. On supprime tous les utilisateurs qui se trouvent dedans et qui n'ont rien a y faire. En gros on laisse le groupe administrators (mon système est en anglais) puis network services (service réseaux en français) sur network services on ne mets que les droit en lecture, on va dans avancé et on coche remplacer les permissions des enfants (si vous n'arrivez pas a supprimer des utilisateurs du répertoire du site c'est que l'heritage des droits est mis en place, dans ce cas dans avancés décochez autoriser l'héritage des permissions). Ensuite pour file on désactive l'héritage des permissions, on ajoute écriture et modification à network services et on remplace les permissions des enfants.

Voilà on a limité les degats au niveau du système de fichier. L'utilisateur ne pourra plus modifier le fichier Web.Config. Le problème c'est qu'il pourra toujours le lire. On va donc mettre en place le CAS. Pour cela on va indiquer en code que l'on veut un accès total sur le répertoire de l'utilisateur, un accès en écriture sur le répertoire file, mais que les répertoires du dessus on ne veut aucun accès. Voici le code.

```

private void btnOK_Click(object sender, System.EventArgs e)
{
    if(!File.Exists(_objUser.Chemin +"\\\" + txtOther.Text))
    {
        StreamWriter objWriter=File.CreateText(_objUser.Chemin +"\\\" +
txtOther.Text);
        objWriter.Flush();
        objWriter.Close();
        ddlFile.DataSource=_objUser.Fichiers;
        ddlFile.DataBind();
        ddlFile.Items.Insert(0,"Nom du fichier");
        ddlFile.Items.FindByText(txtOther.Text).Selected=true;
    }
    else
    {
        PermissionSet per=new PermissionSet(PermissionState.None);
        FileIOPermission objPer=new
FileIOPermission(FileIOPermissionAccess.NoAccess,Server.MapPath("."));

        objPer.AddPathList(FileIOPermissionAccess.AllAccess,_objUser.Chemin);
        per.AddPermission(objPer);
        per.PermitOnly();
        StreamReader objStream= new StreamReader(_objUser.Chemin +"\\\" +
txtOther.Text);
        txtContent.Text =objStream.ReadToEnd();
        objStream.Close();
    }
}

```

Regardons ce code. On crée un objet FileIOPermission qui ne donne aucun accès au repertoire racine et ses repertoire enfants. On ajoute une permission pour le répertoire de l'utilisateur qui est l'accès en contrôle total. Le contexte de sécurité est donc posé. Le code qui sera exécuté en dessous s'exécutera donc dans ce contexte la. Une exception de sécurité sera donc déclenchée si je tente d'accéder a un autre répertoire que mon répertoire utilisateurs.

Maintenant on va sécuriser le code en utilisant une autre technique d'accès aux fichiers, on va

verifier que ce qui est récupéré comme chemin d'accès équivaut au chemin de l'utilisateur et pas un autre chemin pour resituer le code je remet toute la fonction:

```
private void btnOK_Click(object sender, System.EventArgs e)
{
    if(!File.Exists(_objUser.Chemin + "\\\" + txtOther.Text))
    {
        StreamWriter objWriter=File.CreateText(_objUser.Chemin + "\\\" +
txtOther.Text);
        objWriter.Flush();
        objWriter.Close();
        ddlFile.DataSource=_objUser.Fichiers;
        ddlFile.DataBind();
        ddlFile.Items.Insert(0,"Nom du fichier");
        ddlFile.Items.FindByText(txtOther.Text).Selected=true;
    }
    else
    {
        PermissionSet per=new PermissionSet(PermissionState.None);
        FileIOPermission objPer=new
FileIOPermission(FileIOPermissionAccess.NoAccess,Server.MapPath("."));

        objPer.AddPathList(FileIOPermissionAccess.AllAccess,_objUser.Chemin);
        per.AddPermission(objPer);
        per.PermitOnly();
        string strFilePath=Path.GetFullPath(_objUser.Chemin + "\\\" +
txtOther.Text);
        StreamReader objStream;
        if(strFilePath.StartsWith(_objUser.Chemin ))
            objStream= new StreamReader(strFilePath);
        else
            return;
        txtContent.Text =objStream.ReadToEnd();
        objStream.Close();
    }
}
```

Très bien maintenant on va vérifier ce que l'utilisateur a écrit avec une expression régulière et un RegularExpressionValidator qui validera qu'il n'y ait que des caractere alphanumerique de tapé et qu'il n'y ait qu'un point dans la chaine sans autre caractère spéciaux. On validera aussi le fait qu'il y a au maximum 4 caracteres dans l'extension du fichier. On n'oubliera pas de mettre le page.isvalid dans la fonction de recherche.

```
<asp:RegularExpressionValidator id="RegularExpressionValidator2" runat="server"
ValidationExpression="\w{2,}.{0,1}\w{3,4}"
ControlToValidate="txtOther" ErrorMessage="Nom de fichier non
valid"/>

private void btnOK_Click(object sender, System.EventArgs e)
{
    if(!Page.IsValid)
        return;
}
```

Sécurisons la taille des fichiers uploader. Pour cela on va indiquer à .Net que la taille maximal des requêtes qui lui sont envoyé doit être de 2MO. On indique donc dans le fichier Web.Config le bout de code suivant:

```
<location path="file.aspx">
    <system.web>
        <httpRuntime maxRequestLength="2048"/>
    </system.web>
</location>
```

Cette commande permet de couper la connexion si un fichier d'une trop grosse taille est envoyé au serveur. Si bien qu'on n'a pas d'utilisation abusive de la bande passant ce qui limitera les risque de Denial Of Service. Ensuite on ne veut que des fichiers textes (qui ne pourront pas être exécuter sur le serveur) pour cela on va donc faire deux chose, la première c'est de verifier côté serveur qu'on a bien un fichier texte en regardant le mime type et en sauvegardant le fichier avec un objet indépendant du fichier posté. Ensuite on va verifier côté client que l'extension du fichier est .txt.

```
private void btnFileUpload_Click(object sender, System.EventArgs e)
{
    if(!Page.IsValid)
```

```

        return;
    if(fileUpload.PostedFile !=null && fileUpload.PostedFile.ContentLength>0)
    {
        StreamWriter objWrite=new StreamWriter(_objUser.Chemin +"\\\" +new
FileInfo(fileUpload.PostedFile.FileName).Name, false, new
System.Text.UTF8Encoding());
        objWrite.Write(new
StreamReader(fileUpload.PostedFile.InputStream).ReadToEnd());
    }
    ddlFile.DataSource=_objUser.Fichiers;
    ddlFile.DataBind();
    ddlFile.Items.Insert(0,"Nom du fichier");
}

```

Voilà on a sécurisé au maximum la page des fichiers. Maintenant il nous reste un peu de travail sur le site avant de passer aux technologies satellites **Finissons de sécuriser le site**

#### **Commençons par l'erreur dans le global.asax**

Le problème du global.asax est qu'un objet de session est chargé en mémoire avant que l'utilisateur ne s'authentifie. Il faut donc que l'utilisateur soit authentifié avant chaque mise en variable de session d'objet qui lui sont affecté. Pour cela on va changer le code. Il faut savoir que dans mon exemple le fait d'initialiser un objet de session sur le session start n'a pas lieu d'être, je l'ai fait pour illustrer ce type d'attaques.

```

    if(_Mdp==txtPassword.Text)
    {
        System.Web.Security.FormsAuthentication.RedirectFromLoginPage(txtLogin.Text
, false);
        CUser objUser=new CUser(txtLogin.Text);
        Session["objUser"]=objUser;
    }

```

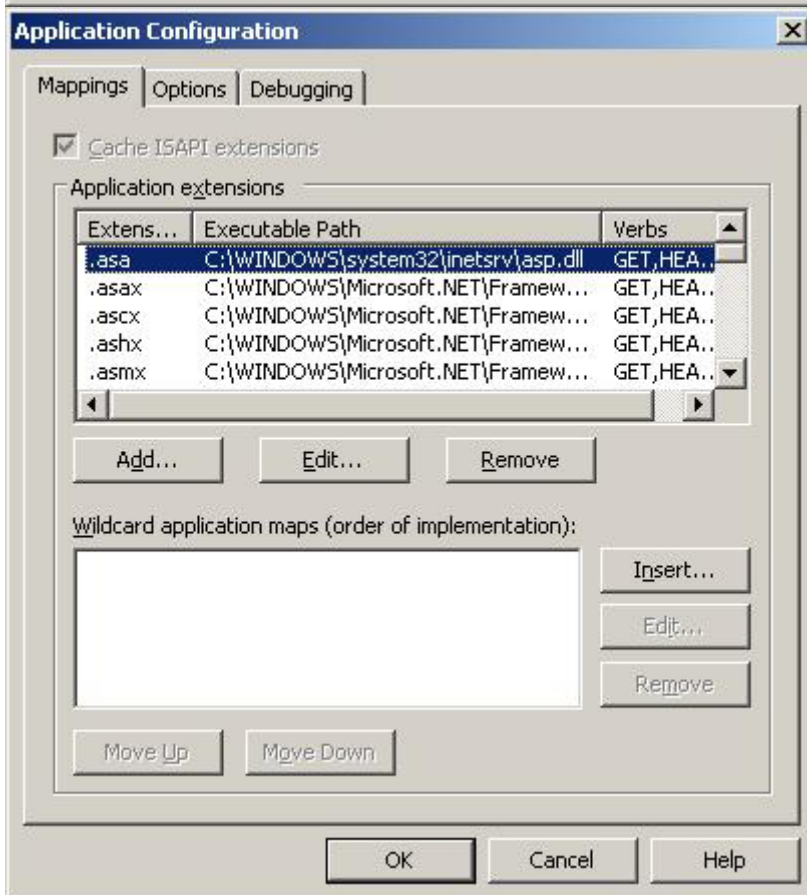
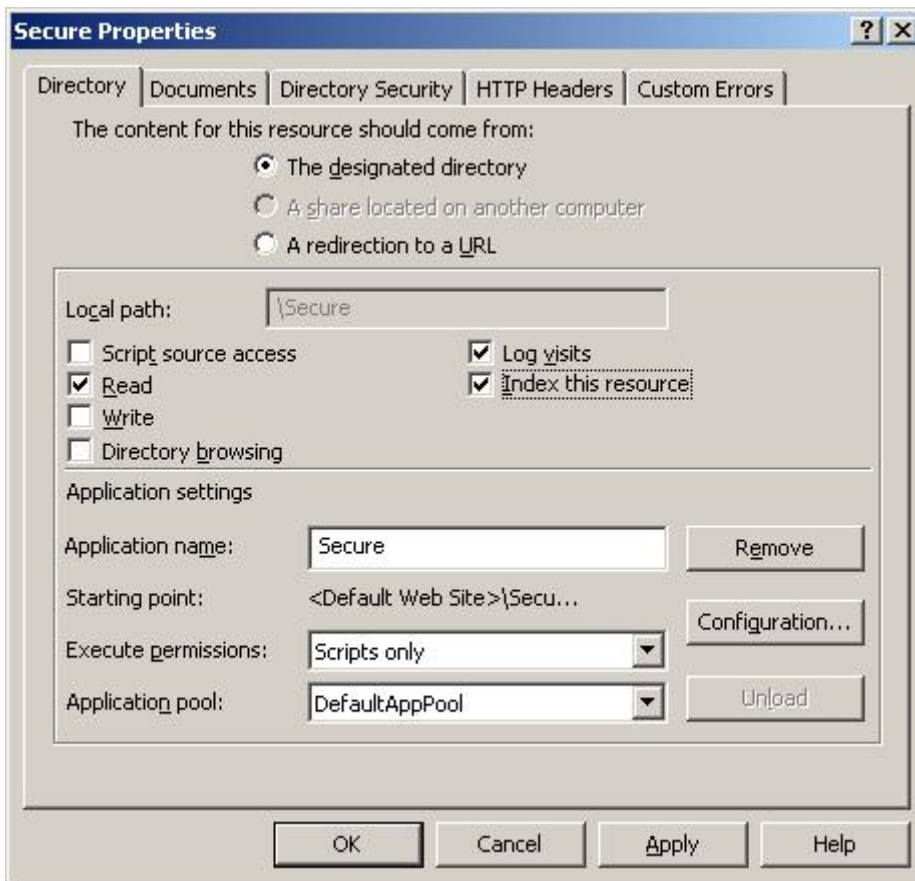
On retire le code qu'il y a dans le session Start et voilà on a une meilleur gestion de la mémoire.

#### **La page Information.htm**

Pour que la page Information.html nécessite une authentification quand on la demande il faut qu'elle soit soumise à l'isapi d'asp.net et que ça ne soit plus IIS qui la gère.

Pour cela on va aller dans la console d'administration de IIS et on va ajouter l'extension HTML à la liste des extensions de l'isapi ASP.NET.

Allons donc dans les outils d'administration, gestionnaire de service internet. La on ouvre les propriétés de notre répertoire virtuel. Dans l'onglet Directory on clique sur le bouton Configuration.



Dans cette page on ajoute une extension d'application avec l'extension HTM qui pointe sur l'exécutable C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322\aspnet\_isapi.dll évidemment à

changer suivant votre configuration.

Et voilà la page est protégée par ASP.Net. La personne devra se loguer pour pouvoir voir les informations de la page information.htm. On aurait pu aussi changer l'extension de la page en .ASPX mais ça aurait été trop facile ;-)

### Les fichiers TXT

Le principe des fichiers TXT est le même que celui de la page information.htm mais avec quelque petite chose en plus.

Commençons par ajouter l'extension txt au moteur ASP.NET comme on a fait avec la page Information.

Maintenant les fichiers TXT sont protégés par mot de passe. Le problème c'est que ce n'est pas ce qu'on souhaite. Ce qu'on veut c'est qu'il soit tout simplement inaccessible.

Pour cela on va ajouter une section dans le fichier Web.config qui interdira les fichiers txt.

```
<httpHandlers>
  <add verb="*" path="*.txt" type="System.Web.HttpForbiddenHandler" />
</httpHandlers>
```

Voilà les fichiers texte sont soumis aux mêmes règles que le fichiers .config c'est à dire qu'ils ne sont pas accessible sur le web.

### Le Webservice

Comme vous l'avez peut être vu, il y' a un webservices dans notre site. Ce Webservice vous permet de récupérer les informations des utilisateurs par rapport au login qui est passer.

Pour sécuriser ce webService on va faire appel a WS-Security. WS-Security se trouve dans WSE (WebService Enhancement) et permet de sécurisé à partir de la norme WSE les services WEB.

Commençons par installer WSE sur le serveur, on va l'installer en mode administrateur.

Maintenant on va lancer l'utilitaire de configuration de service web.

On va ouvrir notre fichier de configuration et mettre en place WSE. ce qui va ajouter ce code au fichier Web.config

```
<webServices>
  <soapExtensionTypes>
    <add type="Microsoft.Web.Services2.WebServicesExtension,
Microsoft.Web.Services2, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" priority="1" group="0" />
  </soapExtensionTypes>
</webServices>
ET
<configSections>
  <section name="microsoft.web.services2"

    type="Microsoft.Web.Services2.Configuration.WebServicesConfiguration,
Microsoft.Web.Services2, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"/>
</configSections>
```

Ensuite nous allons modifier le code du Web Service afin qu'il fasse une authentification par credential passé dans l'entête soap. Pour cela on va utiliser un UserToken qui fait parti de l'espace de nom Microsoft.Web.Services2.Security.Tokens livré avec WSE

```
public CUser GetUserByLogin(string Login)
{
    string Username = string.Empty;
    string Password = string.Empty;
    SoapContext MysoapContext = SoapContext.Current;
    foreach (SecurityToken Mytok in
MysoapContext.Security.Tokens)
    {
        if (Mytok is UsernameToken)
        {
            Username =
((UsernameToken)Mytok).Username;
            Password=((UsernameToken)Mytok).Password;
            break;
        }
    }
}
```

```

        if (Username == string.Empty)
            throw new Exception("No username");
        if (!(Username=="User" && Password=="Pwd"))
        {
            throw new Exception("Invalid credential");
        }
        return new CUser(Login);
    }
}

```

Le web Service marche maintenant par credential. Le client sera obligé d'envoyer les credential par entête soap pour pouvoir utiliser le webservice

## Les techniques de défenses Le Code Access Security

Le CAS permet aux développeurs et aux administrateurs de mettre en place un controle d'accès aux ressources du code managé. Cette sécurité passe par un ensemble de sous technologie que sont l'evidence, les security policy et les code group

### Evidence

L'evidence permet d'identifier l'assembly et de savoir d'ou elle provient afin de lui affecter une security policy. Les information inclus dans l'evidence sont: Son strong name, l'url ou unc d'ou provient l'assembly, le site, la zone de sécurité ou elle se trouve (internet, intranet, ou systeme de fichier local) et l'éditeur de l'assembly.

Ainsi suivant les zones de confiances une security policy lui sera appliquée lui donnant ou non accès a des ressources sur la machine.

### Security policy

Suivant l'evidence de l'assembly une des politique de sécurité est applique a l'assembly:

- Nothing (l'assembly vient d'une zone sensible et pas du tout sur, elle n'a aucun droit sur la machine)
- Execution (l'assembly a juste le droit de s'exécuter mais aucun accès a des ressources sensibles)
- Internet (l'assembly a les même droits que les pages web)
- LocalIntranet (l'assembly a le droit d'execute, de créer des interfaces utilisateurs, d'utiliser les stockages isolés de l'utilisateurs "dans document and settings", de lire le nom de l'utilisateur et les répertoire temporaire.
- Everything (l'assembly a accès a tout excepter le droit de passer outre la verification)
- FullTrust (accès à tout)

### Code group

Les groupes de code sont la pour faciliter l'administration et rassembler de manière logique du code dans des groupes sur lequel on a déjà mis des règles de sécurité.

Voici des exemples de code group:

- Le répertoire de l'application
- l'éditeur de l'assembly
- L'url de provenance de l'assembly
- La zone de sécurité

### Security policy level

Les niveaux de sécurité sont les suivants:

- Entreprise: L'administrateur de l'entreprise définit une sécurité qui s'appliquera à toute la société.
- Machine: securité au niveau de l'ordinateur
- User: sécurité utilisateur
- Application Domain: La seul partie non administrable qui determine la sécurité au niveau de l'application

Pour configurer la sécurité au niveau de l'administration de la machine il existe deux outils, mscorcfg.msc qui se trouve dans les outils d'administration et CasPol qui est un outil en ligne de commande.

Voila la sécurité au niveau du remoting et des webservice ainsi que l'utilisation des outils de configuration sera développé dans la partie 4 de l'article.

**VANNESTE Xavier**



L'ensemble ou partie de ce document ainsi que le code mis à disposition, ne peut être diffusé ailleurs sans autorisation préalable

[elise.dupont@europe.com](mailto:elise.dupont@europe.com) - [www.dotnet-tech.com](http://www.dotnet-tech.com) - 2003-2005