

Areas

ASP.NET MVC

Areas

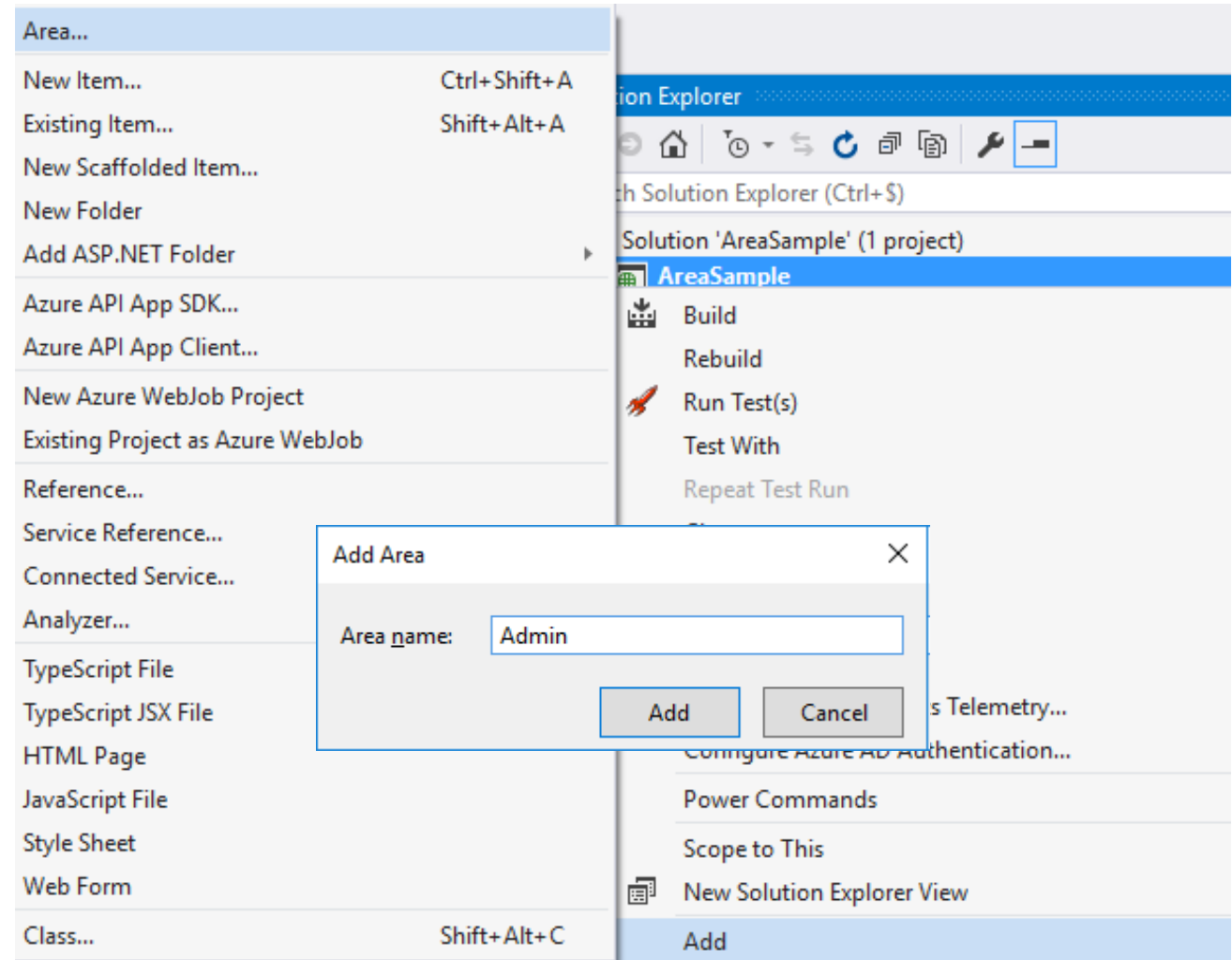
Les zones (Area) ont été introduites en ASP.NET MVC 2 et permettent de séparer nos Modèles, Contrôleurs et vues en « section » fonctionnelles.

Par exemple, pour une zone d'administration dans notre application Web.

La séparation « fonctionnelle » nous permet de faciliter la maintenance de nos applications.

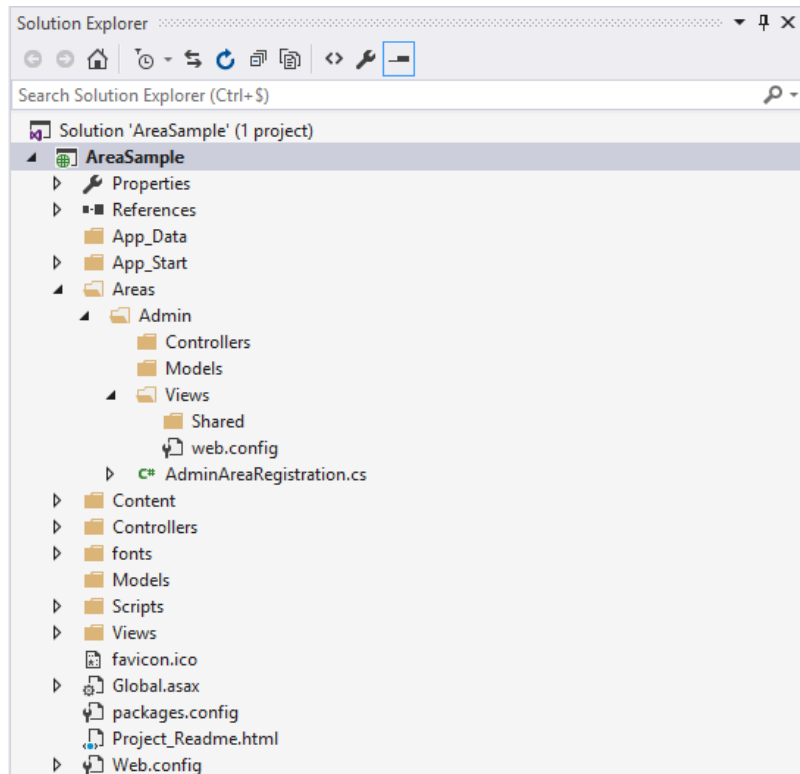
L'enregistrement d'une zone se fait en plusieurs étapes.

La première consistant à ajouter cette zone à notre projet via un Click droit → Add Item → Area sur notre projet.



Areas

Nous obtenons une structure MVC dans un sous-dossier portant le nom de notre area.



Un fichier *nomAreaAreaRegistration.cs* est également créé et contient la définition des routes pour l'area.

```
using System.Web.Mvc;

namespace AreaSample.Areas.Admin
{
    public class AdminAreaRegistration : AreaRegistration
    {
        public override string AreaName
        {
            get
            {
                return "Admin";
            }
        }

        public override void RegisterArea(AreaRegistrationContext context)
        {
            context.MapRoute(
                "Admin_default",
                "Admin/{controller}/{action}/{id}",
                new { action = "Index", id = UrlParameter.Optional }
            );
        }
    }
}
```

Areas

Conflits avec les routes

Si nous avons deux contrôleurs portant le même nom dans l'area et le site principal, le moteur MVC ne pourra pas les distinguer et nous obtiendrons le message d'erreur ci-après.

Afin de remédier à ce problème, nous devons éditer le fichier *RouteConfig.cs* se trouvant dans le dossier *app_start* et ajouter le namespace du contrôleur par défaut que nous désirons utiliser en cas de conflit.



Area

The diagram illustrates the relationship between two namespaces in an ASP.NET MVC application. On the left, the `AreaSample.Controllers` namespace contains the `HomeController` class. On the right, the `AreaSample` namespace contains the `RouteConfig` class. A red box highlights the `AreaSample.Controllers` namespace in the left code block. A red box highlights the `"AreaSample.Controllers"` string in the `namespaces` array of the `RegisterRoutes` method in the right code block. A red line connects the two boxes, indicating that the routing configuration is set up to recognize the `AreaSample.Controllers` namespace.

```
using System.Web.Mvc;

namespace AreaSample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult About()
        {
            ViewBag.Message = "Your application description page.";
            return View();
        }

        public ActionResult Contact()
        {
        }
    }
}

using System.Web.Routing;

namespace AreaSample
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional },
                namespaces: new string[] { "AreaSample.Controllers" }
            );
        }
    }
}
```

Area

Si vous utilisez l'attribut route, vous devez ajouter l'attribut *RouteArea*.

Cet attribut contiendra le nom de l'area et donc pas besoin de le spécifier au sein de l'attribut route.

Il est également possible de changer le préfix. En effet par défaut si l'area s'appelle admin, la route commencera par *admin/....* Si vous désirez changer le préfixe, utilisez la propriété *AreaPrefix* de l'attribut.

```
namespace AreaSample.Areas.Admin.Controllers

[RouteArea("Admin", AreaPrefix = "Manage")]
public class HomeController : Controller
{
    // GET: Admin/Home

    [Route("")]
    [Route("home")]
    [Route("home/index")]
    public ActionResult Index()
    {
        return View();
    }
}
```

Security

ASP.NET MVC

MVC Input et Output

Security – Asp.Net MVC

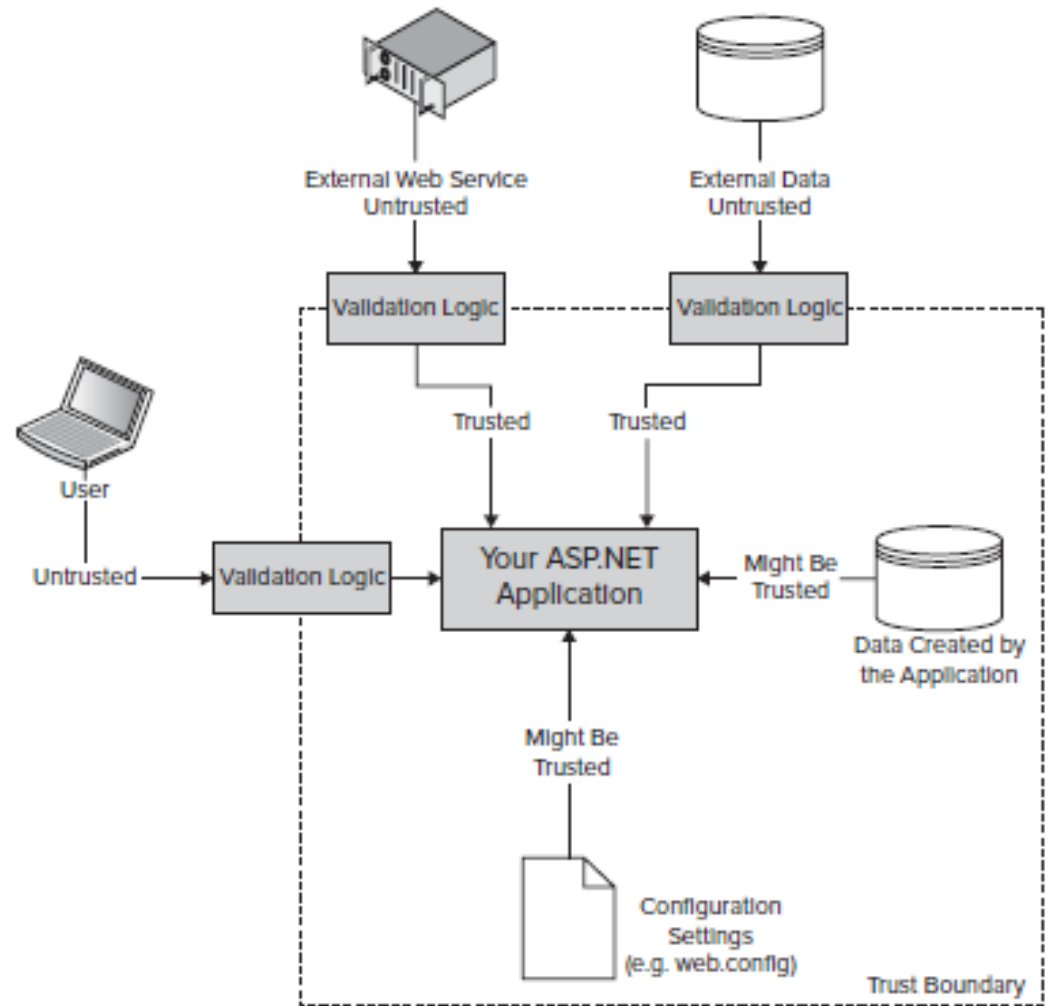
MVC Input et Output

Comment se protéger des attaques XSS (Cross Site Scripting)?

Tous les formulaires sont potentiellement vulnérables aux attaques XSS.

Ces attaques consistent à injecter du javascript via les inputs (formulaires, services , etc...) afin de rediriger les utilisateurs, changer le contenu de la page ou encore voler les infos stockées en session.

Le principe de base est de valider toutes les entrées dans notre zone de confiance



MVC Input et Output

Asp.Net MVC protège déjà certaines injections.

Not Vulnerable to XSS

Votre commentaire :

Valider



MVC Input et Output

Cependant dans certains cas, nous avons besoin d'autoriser les tags HTML....

Nous pouvons dans ce cas, utiliser l'attribut `[ValidateInput(false)]` sur l'action du controller qui devra accepter les tags...

En faisant cela, nous acceptons donc directement le js ou l'html,... en input pour notre action...

Lorsque je veux ensuite afficher la donnée, ASP MVC encode le résultat en html grâce à l'@.

Si j'ai cependant besoin d'afficher avec formatage html, je devrais utiliser `@Html.Raw(...)`

The image shows a Visual Studio environment with a C# controller and a Razor view. The controller has two actions: `Index()` and `VulnerableXSS()`. The `VulnerableXSS()` action is annotated with `[HttpPost]` and `[ValidateInput(false)]`. The `VulnerableXSS(string Comment)` action sets `ViewBag.Comment = Comment;` and returns `View()`. The view contains the following HTML:

```
<h2>Vulnerable to XSS</h2>
<form method="post">
  <b><u>Votre commentaire :</u></b><br />
  @Html.EditorFor(M=>M.Comment)
  <input type="submit" class="btn btn-info"/>
  @Html.Raw(ViewBag.Comment)
</form>
```

Annotations 1-4 point to the following elements:

- 1: `public ActionResult Index()`
- 2: `[ValidateInput(false)]`
- 3: `ViewBag.Comment = Comment;`
- 4: `@Html.Raw(ViewBag.Comment)`

On the right, a Chrome error message is displayed:

Cette page ne fonctionne pas

Chrome a détecté un code inhabituel sur cette page et a bloqué cette dernière pour protéger vos informations personnelles (mots de passe, numéros de téléphone et de cartes de paiement).

Essayez de [consulter la page d'accueil du site.](#)

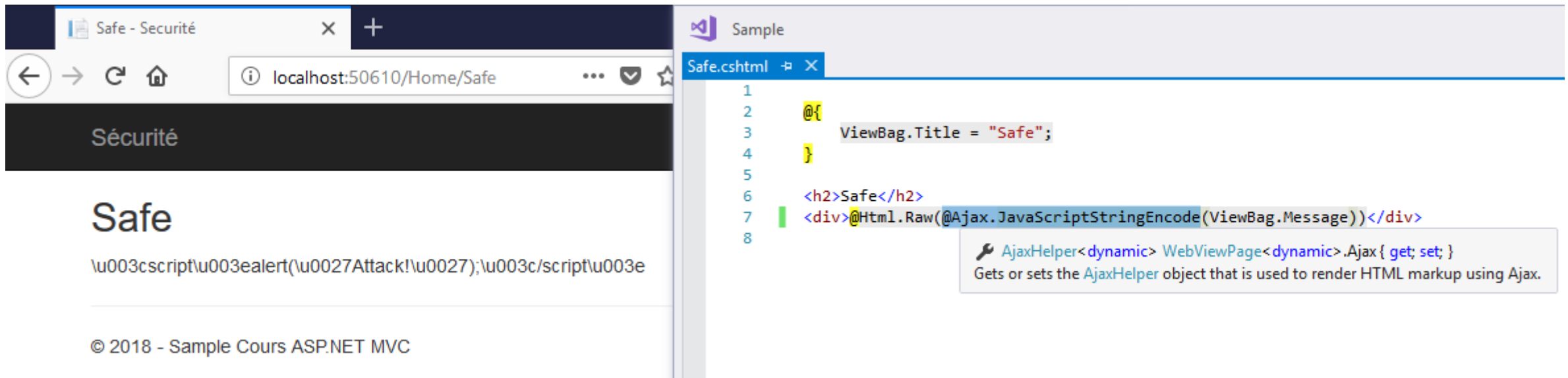
ERR_BLOCKED_BY_XSS_AUDITOR

MVC Input et Output

Si nous voulons que seul une propriété permette l'encodage : [AllowHtml]

Nous devons également prêter attention aux paramètres de l'url, aux paramètres renvoyés par un services, une DB,...

Si l'helper `Html.Raw` peut nous éviter des surprise, nous pouvons également utiliser `@Ajax.JavaScriptStringEncode`



The screenshot shows a web browser window on the left and a Visual Studio code editor on the right. The browser window displays a security warning in French: "Sécurité" (Security) and "Safe". Below the warning, the text "Safe" is visible, followed by a JavaScript alert message: `\u003cscript\u003ealert(\u0027Attack!\u0027);\u003c/script\u003e`. The Visual Studio code editor shows a file named `Safe.cshtml` with the following code:

```
1
2  @{
3      ViewBag.Title = "Safe";
4  }
5
6  <h2>Safe</h2>
7  <div>@Html.Raw(@Ajax.JavaScriptStringEncode(ViewBag.Message))</div>
8
```

A tooltip for the `AjaxHelper` property is visible, stating: "Gets or sets the `AjaxHelper` object that is used to render HTML markup using Ajax."

MVC Input et Output

Microsoft propose également une librairie : *AntiXSS library*.

Vous trouverez de la lecture supplémentaire sur : <http://go.microsoft.com/fwlink/?LinkID=293690&clcid=0x409>

Et pour utiliser la librairie, un simple *using* suffit :

```
@using Microsoft.Security.Application
```

```
<div class="messages">@Encoder.JavaScriptEncode(ViewBag.Msg)</div>
```

MVC Input et Output

Une autre type d'attaque : Cross-Site Request Forgery (CSRF)

Le principe de cette attaque est de détourner une requête afin de se faire passer par le client ayant initié le dialogue avec le serveur.

Exemple :

Vous recevez par mail un lien permettant de , par exemple, voir les résultat d'un concours que vous avez peut-être gagné.

`View Result`

Si vous cliquez sur le lien, l'action de la vue va donc s'exécuter et comme vous êtes un user authentifié, le pirate peut accéder au système via votre compte.

MVC Input et Output

Pour prévenir ce genre d'attaque :

- Assurez vous qu'il n'est pas possible d'utiliser plusieurs fois le lien.
- La méthode GET ne doit permettre que la récupération de données et non la modification
- Assurez-vous que la requête ne peut pas être rejouée si par exemple un pirate utilise Javascript pour simuler un POST

L'html helper `@Html.AntiForgeryToken()` permet d'éviter le replay en fournissant un token unique pour vos requêtes.

Cette fonction utilise un Hidden input mais également un cookie afin de vérifier la provenance et la légitimité de la requête

View

```
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken();
    @Html.EditorForModel();
    <input type="submit" value="Submit" />
}
```

Action

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Crsf(Person P)
{
    return View();
}
```