# Sécuriser les formulaires avec Symfony

[sommaire]

1. ## Objectifs

- Connaitre les méthodes de sécurisation d'un formulaire avec Symfony.
- Etre capable de sécuriser un formulaire Symfony.

- ## Présentation

- Un formulaire est un vecteur d'attaque potentiel pour une personne malveillante.
- Symfony propose par défaut des mécanismes de sécurité pour protéger contre les attaques telles que le CSRF (Cross-Site Request Forgery).
- Dans ce formulaire, nous allons voir comment protéger très facilement une application Symfony contre les attaques **CSRF** (Cross Site Request Forgery).
- La forme complète de CSRF est la falsification de requêtes intersites. Il s'agit d'un type d'attaque en ligne dans lequel l'attaquant envoie des demandes en tant qu'utilisateur autorisé à un système en obtenant des informations d'accès d'un utilisateur particulier de ce système et effectue différents types d'activités malveillantes en utilisant l'identité de cet utilisateur. L'impact de cette attaque dépend des privilèges de la victime sur le système.
- Si la victime est un utilisateur normal, cela n'affectera que les données personnelles de la victime. Mais si la victime est l'administrateur du système, l'attaquant peut endommager l'ensemble du système.

- ## Fonctionnement du CSRF

- La falsification de requêtes intersites (CSRF) est un type d'attaque effectué par l'attaquant pour envoyer des requêtes à un système avec l'aide d'un utilisateur autorisé auquel le système fait confiance.
- Symfony fournit une protection contre les attaques CSRF en générant un jeton CSRF. Ce jeton CSRF est généré automatiquement pour chaque utilisateur. Ce jeton est utilisé pour vérifier les demandes des utilisateurs et est automatiquement inclus dans les formulaires générés par Symfony.
- Cette protection CSRF est automatiquement appliquée à tous les formulaires HTML dans l'application Symfony. Symfony inclut un mécanisme qui ajoute un jeton CSRF automatiquement à chaque formulaire généré, ce qui protège contre les attaques CSRF.

apcpedagogie.com

- Une des manières efficaces de se prémunir contre les attaques CSRF est d'accompagner chaque formulaire à protéger d'un jeton d’identification de session. Symfony gère cela automatiquement en incluant un jeton CSRF dans chaque formulaire généré.
- Lorsque le formulaire est soumis, Symfony vérifie automatiquement que le jeton CSRF inclus dans la soumission correspond à celui généré pour la session en cours. Si les jetons ne correspondent pas, Symfony arrête le traitement du formulaire et empêche la soumission.

## La mise en œuvre

- Pour sécuriser les formulaires avec Symfony, vous pouvez suivre ces étapes :

1. **Utiliser les formulaires Symfony :**

- Utilisez le composant de formulaire intégré de Symfony pour créer et gérer vos formulaires. Les formulaires Symfony intègrent automatiquement des fonctionnalités de sécurité telles que la protection CSRF.

- **Activer la protection CSRF :**

- La protection CSRF est activée par défaut dans Symfony. Cela signifie que chaque fois que vous créez un formulaire avec Symfony, un jeton CSRF unique est automatiquement généré et inclus dans ce formulaire.
- Symfony utilise un composant appelé **CsrfTokenManager** pour gérer la génération et la vérification des jetons CSRF. Ce composant est automatiquement intégré dans le processus de création des formulaires, ce qui garantit que la protection CSRF est activée pour tous les formulaires par défaut.
- Ainsi, en utilisant les formulaires Symfony de manière standard, vous bénéficiez automatiquement de la protection CSRF sans avoir à effectuer de configuration supplémentaire. C'est l'un des nombreux avantages de l'utilisation du framework Symfony pour le développement web sécurisé et efficace.

- **Inclure le champ CSRF dans vos formulaires :**

- Symfony inclut automatiquement un champ CSRF dans chaque formulaire généré. Vous n'avez donc pas besoin de vous soucier de l'ajout manuel de ce champ.
- Symfony inclut automatiquement un champ CSRF dans chaque formulaire généré. Ce champ est utilisé pour stocker le jeton CSRF unique généré par Symfony pour ce formulaire

apcpedagogie.com

spécifique. Vous n'avez donc pas besoin de vous soucier de l'ajout manuel de ce champ dans vos formulaires.

- Lorsque vous créez un formulaire avec Symfony, le champ CSRF est automatiquement inclus dans le formulaire par le composant de formulaire Symfony. Ce champ est généralement rendu sous forme de champ de type "hidden" dans le HTML généré.
- Voici un exemple de rendu d'un formulaire avec Symfony, où le champ CSRF est automatiquement inclus :

```
{# templates/example/index.html.twig #}
{% extends 'base.html.twig' %}

{% block body %}

<h1>Example Form</h1>

    {{ form_start(form) }}
    {{ form_widget(form) }}
    {{ form_end(form) }}
{% endblock %}
```

- Dans cet exemple, lorsque vous utilisez la fonction form_start(form) pour démarrer votre formulaire, Symfony inclut automatiquement le champ CSRF dans le formulaire. Vous n'avez donc pas besoin de vous soucier de son ajout manuel.
- Cela simplifie grandement le processus de sécurisation des formulaires contre les attaques CSRF, car vous n'avez pas à vous soucier de la gestion manuelle des jetons CSRF. Symfony le gère pour vous automatiquement.

- **Valider les données du formulaire :**
  - Utilisez les validateurs intégrés de Symfony pour valider les données soumises par les utilisateurs. Cela garantit que seules les données valides sont acceptées et traitées par votre application.
  - Symfony propose une série de validateurs intégrés que vous pouvez utiliser pour valider les données soumises par les utilisateurs. Ces validateurs permettent de garantir que seules les données valides sont acceptées et traitées par votre application, ce qui contribue à la sécurité et à la fiabilité de votre application.

apcpedagogie.com

- Voici comment vous pouvez utiliser les validateurs intégrés de Symfony dans votre formulaire pour valider les données :

```
// src/Form/ExampleFormType.php
namespace App\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Validator\Constraints\NotBlank;

class ExampleFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('name', TextType::class, [
                'constraints' => [
                    new NotBlank(['message' => 'Veuillez entrer un nom.'])
                    // Ajoutez d'autres validateurs selon vos besoins
                ]
            ])
            ->add('submit', SubmitType::class)
        ;
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            // Configurez vos options de formulaire ici
        ]);
    }
}
```

- Dans cet exemple, nous utilisons le validateur NotBlank pour s'assurer que le champ "name" du formulaire n'est pas vide. Vous pouvez ajouter d'autres validateurs selon vos besoins pour valider d'autres aspects des données soumises.
- Lorsque le formulaire est soumis, Symfony applique automatiquement ces validateurs aux données soumises. Si une validation échoue, Symfony renvoie les messages d'erreur appropriés à l'utilisateur pour l'aider à corriger les problèmes.

apcpedagogie.com

- En utilisant les validateurs intégrés de Symfony, vous pouvez garantir que seules les données valides sont acceptées et traitées par votre application, ce qui contribue à sa sécurité et à sa fiabilité.

- **Gérer les erreurs de validation :**
  - En cas d'erreurs de validation, assurez-vous de renvoyer les messages d'erreur appropriés à l'utilisateur pour l'aider à corriger les problèmes.
  - Voici un exemple simple de création et de traitement d'un formulaire Symfony :

```
// src/Form/ExampleFormType.php
namespace App\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

class ExampleFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('name', TextType::class)
            ->add('submit', SubmitType::class)
        ;
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            // Configure your form options here
        ]);
    }
}
// src/Controller/ExampleController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;
```

apcpedagogie.com

```
use App\Form\ExampleFormType;

class ExampleController extends AbstractController
{
    /**
     * @Route("/example", name="example")
     */
    public function index(Request $request)
    {
        $form = $this->createForm(ExampleFormType::class);

        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            // Traitez les données du formulaire ici
        }

        return $this->render('example/index.html.twig', [
            'form' => $form->createView(),
        ]);
    }
}

{# templates/example/index.html.twig #}
{% extends 'base.html.twig' %}

{% block body %}

# Example Form


    {{ form_start(form) }}
    {{ form_widget(form) }}
    {{ form_end(form) }}
{% endblock %}
```

- Avec cette approche, Symfony gère automatiquement la génération et la validation du jeton CSRF pour protéger votre formulaire contre les attaques CSRF.

- **Activité**

apcpedagogie.com

- **Créer un formulaire Symfony** : Utilisez le composant Form de Symfony pour créer un formulaire PHP qui gère la saisie des utilisateurs.
- **Utiliser les annotations de Symfony pour la route** : Au lieu de définir l'action du formulaire directement dans le formulaire HTML, utilisez les annotations de Symfony pour définir la route à laquelle le formulaire sera envoyé.
- **Gérer la soumission du formulaire dans un contrôleur Symfony** : Créez un contrôleur Symfony qui gère la soumission du formulaire et le traitement des données soumises.
- Voici comment vous pouvez adapter le code pour Symfony :

```
// src/Form/ContactFormType.php
namespace App\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type\TextareaType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

class ContactFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('prenomNom', TextType::class, ['label' => 'Prénom & Nom'])
            ->add('email', EmailType::class, ['label' => 'Email'])
            ->add('message', TextareaType::class, ['label' => 'Message'])
            ->add('send', SubmitType::class, ['label' => 'Submit'])
        ;
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            // Configurez vos options de formulaire ici
        ]);
    }
}
```

- Le résultat du code ci-dessus est le formulaire ci-dessous que l'utilisateur final peut afficher:

- **Créer le contrôleur Symfony :**

apcpedagogie.com

```
// src/Controller/ContactController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use App\Form\ContactFormType;

class ContactController extends AbstractController
{
    /**
     * @Route("/contact", name="contact")
     */
    public function index(Request $request): Response
    {
        $form = $this->createForm(ContactFormType::class);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $formData = $form->getData();
            // Traitement des données ici
            // Par exemple, rediriger ou envoyer un email

            return $this->redirectToRoute('success');
        }

        return $this->render('contact/index.html.twig', [
            'form' => $form->createView(),
        ]);
    }

    /**
     * @Route("/success", name="success")
     */
    public function success(): Response
    {
        // Page de réussite après l'envoi du formulaire
        return $this->render('contact/success.html.twig');
    }
}
```

- **Créer les vues Twig :**

apcpedagogie.com

```
{# templates/contact/index.html.twig #}
{% extends 'base.html.twig' %}

{% block body %}




{% endblock %}
```

```
{# templates/contact/success.html.twig #}
{% extends 'base.html.twig' %}

{% block body %}

# Formulaire soumis avec succès !

Merci d'avoir soumis le formulaire.

{% endblock %}
```

- Avec cette approche, vous créez un formulaire Symfony en utilisant le composant Form de Symfony et gérez la soumission du formulaire dans un contrôleur Symfony. Cette méthode suit les pratiques recommandées de Symfony pour la création et la gestion des formulaires.

```
public function postContact(Request $request) {
   return $request-> all();
}
```

- La sortie obtenue renverra JSON avec un jeton comme indiqué ci-dessous:

---

apcpedagogie.com

**Sommaire du cours Symfony**

Source : https://apcpedagogie.com/securiser-les-formulaires-avec-symfony · téléchargé le 03/11/2026 · © APC Pédagogie · apcpedagogie.com

apcpedagogie.com