VAADATA
LE BLOG

TECHNIQUE

La sécurité sur Symfony – Episode 1 – injection, xss, auth

7 DÉCEMBRE 2015

Comme beaucoup de frameworks, Symfony comprend des fonctionnalités de sécurité. Symfony est plutôt robuste par défaut, mais ce serait tout de même une erreur de s'appuyer entièrement sur cette réputation pour assurer la sécurité d'une application.
Parce que chaque application est différente, le contexte de sécurité est différent lui aussi. Les développeurs ont donc toujours un rôle important à jouer pour s'assurer que tout est bien configuré et testé.

Filtrage des inputs, authentification, gestion des sessions… Cette série d'articles aborde les différentes mécaniques que tout développeur Symfony peut utiliser, afin d'assurer la sécurité et la fiabilité de l'application qu'il développe.
Et parce que "built-in security" ne signifie pas que tout est déjà fait, nous verrons quelles sont les choses réellement importantes, et les pièges classiques.

Protéger votre application Symfony contre les injections

Si vous ne savez pas ce que sont les injections, vous trouverez des informations utiles dans cet article : Comprendre les vulnérabilités web en 5 min – episode #1 : Injections!

Protéger votre application Symfony contre les injections est assez simple.

Comme pour tout framework ou langage de programmation, la règle numéro #1 est la validation des inputs :
Vous vous attendez à ce qu'un nombre soit fourni par l'utilisateur dans un de vos formulaires ? Vérifiez que la donnée fournie est effectivement un nombre.
Par chance, Symfony propose une façon simple de mettre en place des contraintes, et de créer des contraintes personnalisées.

- Contraintes de validation ("validation constraints") : http://symfony.com/doc/current/reference/constraints.html
- Créer des contraintes personnalisées : http://symfony.com/doc/current/cookbook/validation/custom_constraint.html

TOUTES les inputs doivent être filtrées. Champs de formulaires, paramètres d'URLs, mais également les cookies ou les headers HTTP, si certains d'entre eux sont traités par votre application.
Un enseignement tiré des audits de sécurité que nous avons réalisés sur des applications Symfony est que la plupart des applications procèdent à des validations d'inputs. Mais quelques paramètres passent généralement à la trappe. Malheureusement une erreur suffit pour ouvrir une faille. Une bonne pratique consiste donc à maintenir une liste des inputs utilisés par l'application, et leur contrôles de sécurité correspondants (par exemple dans les spécifications de l'application).

Concernant la base de données, utiliser les mécaniques proposées par Symfony est une bonne pratique.

Si vous utilisez Doctrine (qui est fourni avec la version standard de Symfony), alors vous êtes protégé, à minima pour les requêtes "classiques" sur des objets.
Lors de l'utilisation de DQL pour des requêtes un peu plus personnalisées, vous devez vous assurer que les valeurs provenant d'entrées utilisateurs sont bien communiquées comme des placeholders, avec la méthode setParameter. Ainsi, les requêtes personnalisées deviennent des "requêtes paramétrées", et évitent les injections SQL.
DQL: http://symfony.com/doc/current/book/doctrine.html#querying-for-objects-with-dql

Doctrine n'est pas la seule solution, les autres ORMs comme Propel protègent également des injections SQL, si vous les utilisez correctement.

Si vous envoyez des commandes vers le système d'exploitation (OS commands) ou d'autres requêtes sensibles, vous devez vous assurer que les données passées en paramètres sont bien également "échappées" (input escaping), en fonction du contexte.

Gérer l'authentification et les sessions de manière sécurisée

L'authentification est un sujet assez fourni. Nous n'allons pas parcourir toutes les possibilités offertes par Symfony sur ce plan, le security cookbook est très détaillé sur ce sujet.
Pour vous assurer que votre authentification Symfony est bonne, vous pouvez donc vous référer directement à la section Authentification du cookbook:
http://symfony.com/doc/current/cookbook/security/index.html

Quelques points que vous ne devez pas louper :
- vous assurer que les mots de passe sont hashés dans la base de données (avec bcrypt, par exemple)
- vous assurer que les cookies sont "secure" (secure flag) et "httponly".
Le flag "secure" n'a pas de sens si votre application n'est pas protégée par HTTPS, mais si vous accordez un minimum d'importance aux identifiants de vos utilisateurs, alors HTTPS n'est pas une option.

Ensuite, certains points peuvent être renforcés, en particulier concernant la protection contre les attaques brute-force ou attaques par dictionnaire.
Mettre en place une politique de mots de passe solide est une première stratégie, afin d'éviter que des attaquants trouvent les mots de passe les plus évidents. Trouver le juste équilibre entre commodité et sécurité est le plus difficile !
Si votre application gère des données très sensibles, ajouter un pare-feu applicatif (WAF – Web Application Firewall) est une option à considérer.
En fonction de votre budget et contexte, différentes solutions sont possibles. Certains pare-feux sont capables de détecter et bloquer des attaquants, même si leur adresse IP change.

Protéger une application Symfony contre le Cross Site Scripting (XSS)

Deux règles de base contre le XSS :

1. filtrer les entrées (inputs)
2. échapper les données en sortie (output)

Filtrer les entrées peut être fait avec les contraintes, comme détaillé précédemment dans la section injections.
Vous devez d'abord vous assurer que vous recevez le bon type de données de vos utilisateurs. Mais pour les données "complexes", comme des champs de texte libre (pour un commentaire ou une description, par exemple), l'utilisateur peut fournir presque n'importe quel type de données.
Si vous devez accepter certains caractères spéciaux, ou même des balises html (attention !), alors vous pouvez utiliser des "data transformers" pour implémenter des traitements de données puissants et spécifiques.
Implémenter des data transformers :
http://symfony.com/doc/current/cookbook/form/data_transformers.html

Échapper les données en sortie est une obligation.
Les moteurs de templates comme Twig font un bon travail dans ce domaine, en vous permettant d'échapper les données (avec la syntaxe {{ user.username|escape }} par exemple).
En savoir plus sur ce sujet :
http://symfony.com/doc/current/book/templating.html#output-escaping-in-twig

Les headers Content Security Policy sont aussi des solutions que vous pouvez implémenter sur votre projet, pour donner des instructions claires aux navigateurs web.
Ces headers sont généralement implémentés au niveau du serveur (nginx, httpd…) ou directement dans Symfony avec HttpFoundation.
En savoir plus sur la CSP : https://developer.mozilla.org/en-US/docs/Web/Security/CSP
Et plus généralement sur les headers HTTP relatifs à la sécurité : Sécurisez votre site web avec les headers HTTP

Encore une fois, appliquer ces bonnes pratiques est une bonne chose, mais les appliquer partout dans votre application est un must.

Les prochains articles de cette série traiteront d'autres sujets communs de sécurité avec Symfony.
Vous pouvez nous suivre sur Twitter afin d'être informé des nouveaux articles.

Articles en lien

Audit en boite blanche d'un pipeline CI/CD sur AWS
14 AVRIL 2023

Comment modifier des mots de passe pour sécuriser leur stockage avec Argon2 ?
7 AVRIL 2023

Détournement de session (Hijacking) : principes, types d'attaques et exploitations
7 AVRIL 2023

Langue

Anglais
Français

TOUS NOS ARTICLES

Catégories

Risques
Solutions
Technique

Articles récents

Chiffrement des données et défaillances cryptographiques : Top 10 OWASP #2

Audit en boite blanche d'un pipeline CI/CD sur AWS

Comment modifier des mots de passe pour sécuriser leur stockage avec Argon2 ?

Détournement de session (Hijacking) : principes, types d'attaques et exploitations

Top 10 OWASP #1 : sécurité et vulnérabilités du contrôle d'accès

Rechercher

Tapez et appuyez sur Entrée

Restons connectés !

Recevez des informations sécurité (sélection d'articles, événements, formations ...) max. 1 envoi par mois

Prénom*

E-mail Professionnel*

J'accepte de recevoir des communications de