Filtrage des caractères « dangereux » dans un formulaire

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 février 2011

https://www.bortzmeyer.org/caracteres-dangereux.html

Il y a un débat récurrent à l'intersection du monde de la sécurité informatique et de celui du développement Web sur la meilleure façon de traiter les caractères « dangereux » lorsqu'ils sont entrés par un utilisateur dans un formulaire de saisie. Faut-il interdire ces caractères dangereux, n'autoriser que les caractères sûrs ou bien... ?

Prenons l'exemple d'un formulaire Web où l'utilisateur doit taper son nom. Le problème de départ est que certains caractères sont spéciaux <https://www.bortzmeyer.org/pas-de-caracteres-speciaux.html> pour certains environnements. Ainsi, l'apostrophe ferme les chaînes de caractères en SQL et, sans précautions particulières, une apostrophe dans un nom peut mener à une injection SQL <https://www.bortzmeyer.org/sql-injection.html>, comme illustré dans un dessin fameux de XKCD <http://xkcd.com/327/>. De même, les chevrons sont spéciaux en HTML et peuvent être utilisés pour faire des attaques XSS, si un méchant tape comme nom `<script>do_something();</script>`.

Historiquement, la première défense contre ces attaques avait été d'interdire les caractères « dangereux ». Si les données sont à un moment ou à un autre transmises à un SGBD, on interdit l'apostrophe. Si elles sont à un moment ou à un autre affichées sur une page Web, on interdit les chevrons. Cette méthode est aujourd'hui très critiquée et largement considérée comme peu sûre car il est très difficile d'avoir une liste complète de tous les caractères dangereux pour une application donnée. Une des premières bogues du serveur HTTP Apache était d'interdire la chaîne `..` dans les noms de fichiers demandés (pour empêcher qu'un attaquant ne remonte plus haut que la racine du site Web) en oubliant que ces noms étaient traités par un analyseur qui acceptait les guillemets et laissait donc passer `."."/repertoire/ou/je/veux/aller`... Autre exemple, en SQL, chaque SGBD rajoute quelques caractères spéciaux à la norme SQL. Ainsi, MySQL considère les guillemets comme caractères spéciaux au même titre que l'apostrophe et il faut donc les filtrer également... Et songez que le même nom entré dans un formulaire sera peut-être transmis à un SGBD, sur une page Web, et à un shell Unix. La liste des caractères dangereux devient de plus en plus difficile à établir.

Donc, l'approche « négative » (interdire les caractères dangereux) est très peu sûre. Elle est en outre trop violente dans certains cas. Ainsi, un nom comme « O'Reilly » ou « D'Alembert » est parfaitement légitime et il serait anormal de demander à ces personnes de changer de nom pour éviter les injections

SQL... (Au passage, tout programmeur digne de ce nom devrait avoir lu l'excellent « "*Falsehoods Programmers Believe About Names*" <http://www.kalzumeus.com/2010/06/17/falsehoods-programmers-beli
> » qui donne une idée de la variété des noms possibles pour les humains.)

Qu'en est-il de l'approche « positive », n'autoriser que des caractères sûrs ? Alors que même le plus débutant des bricoleurs PHP sait que l'approche négative est mauvaise, l'approche positive est au contraire celle qui est généralement recommandée. Par exemple, les fanas des expressions rationnelles vont autoriser uniquement les noms correspondant à l'expression `[a-zA-Z]+` puis, après quelques protestations d'utilisateurs nommés Robbe-Grillet ou Du Pont, passer à `[a-zA-Z -]+`. Cette approche est très sûre, car elle ne nécessite pas d'avoir une liste exhaustive des caractères dangereux. Elle est souvent nommée "*input sanitization*" (par exemple dans le dessin de XKCD, qui la conseille implicitement).

Mais elle est aussi très restrictive. Pour les noms de famille pris comme exemple, elle interdirait O'Reilly. Et, si on l'autorise, on se retrouve avec le problème de départ, la présence d'un caractère dangereux dans le nom. Les seuls cas où cette méthode positive marche bien, ce sont ceux où les valeurs à entrer dans le formulaire sont contraintes par des règles syntaxiques strictes. Par exemple, s'il s'agit, non plus de noms quelconques mais d'identificateurs formatés selon une certaine grammaire, alors la méthode positive est acceptable, à condition que le programmeur suive réellement la grammaire en question. Or, bien des programmeurs ne prennent jamais la peine de lire les spécifications et font donc du code qui restreint arbitrairement les termes possibles, par exemple dans les adresses de courrier <https://www.bortzmeyer.org/arreter-d-interdire-des-adresses-legales.html>.

En pratique, la méthode positive se heurte donc vite à des limites. C'est encore plus net si on prend en compte Unicode. Car, si on a des utilisateurs nommés Sk[Caractère Unicode non montré [1]]odowska ou Liétard, ASCII ne suffit plus. On peut passer aux expressions Unicode <http://unicode.org/reports/tr18/> (si le moteur d'expressions rationnelles que vous utilisez les gèrent, ce qui, en 2011, devrait être la règle) et on écrit quelque chose comme `[\p{L}\p{Zs}-']+` (où `p{L}` est l'ensemble des lettres). Mais la liste des caractères ayant la propriété L (pour « lettre ») est tellement longue (la grande majorité des caractères Unicode sont des lettres, plus de 100 000 dans la version 6) qu'il serait imprudent de jurer qu'il n'y en a pas un dans le lot qui soit dangereux dans certaines circonstances... (Déjà, dans mon expression, il y a l'apostrophe.)

Que reste-t-il comme solution ? Elle est plus difficile et peut dans certains cas être moins sûre mais la seule méthode correcte est de ne pas filtrer en **entrée** mais en **sortie**. À part pour les cas où on peut s'appuyer sur une norme précise pour définir un jeu de caractères autorisé **et** où ce jeu ne comporte aucun caractère dangereux pour aucun des logiciels qu'on utilisera, c'est la bonne approche. Il faut donc accepter tout ce qui correspond à la sémantique du champ en question, puis contrôler l'utilisation de ces données au moment de l'exportation (vers SQL, vers PDF, vers LaTeX, vers HTML, etc). C'est une simple application du **principe de robustesse** (« soyez tolérant dans ce que vous acceptez et prudent dans ce que vous envoyez »).

Pour faciliter la tâche du programmeur, et pour diminuer les risques d'oubli, il est bien sûr recommandé de se servir pour ce contrôle de mécanismes de gabarits comme les requêtes préparées en SQL ou comme des gabarits XML, par exemple TAL <https://www.bortzmeyer.org/generer-html-avec-tal.html>.

Merci à Pierre Beyssac, Ollivier Robert et Bertrand Petit pour leurs conseils avisés.

1. Car trop difficile à faire afficher par LaTeX