SpaceFox

Zeste de savoir

Protéger simplement un formulaire contre le spam en 2020

9 septembre 2020

Table des matières

Il se trouve que j'ai dû récemment mettre en place une protection anti-spam sur un formulaire (parce que les dizaines de messages en russe par jour, bof), et que j'ai dû revoir mon panel de solutions sur ce qui fonctionne et qui ne fonctionne pas.

1. Cas d'utilisation

Le but du jeu ici de protéger **un simple formulaire** (type formulaire de contact et de commentaire) contre le spam de type «publicitaire» générique.

Il n'est pas question de se protéger contre la création de comptes multiples, contre une tentative de déni de service ou contre une attaque personnalisé – tout ça demande des moyens beaucoup plus complexes.

D'autre part, je veux **respecter l'utilisateur** et ça implique deux choses:

1. Éviter de lui rendre pénible l'utilisation du formulaire en imposant des actions inutiles et chiantes comme «Déchiffrer des lettres distordues» ou «cliquer sur les passages piétons».
2. Éviter de lui faire charger douze tonnes de JS et d'envoyer toutes ses données dans la nature pour un simple formulaire.

Et enfin, je veux me simplifier la vie, en évitant d'avoir à créer des comptes, configurer des machins, installer des trucs spécifiques compliqués sur mes serveurs et dépendre d'acteurs tiers qui peuvent me couper le service (volontairement ou non) ou le rendre payant.

2. Ce qui ne fonctionne plus

Il y a encore quelques années, les robots spammeurs étaient idiots: ils lisaient le HTML de la page, détectaient le formulaire, remplissaient tout ce qu'ils pouvaient et envoyaient le tout. Il suffisait de créer un *honeypot* («pot de miel») en l'existence d'un champ caché. Si le formulaire était renvoyé avec ce champ rempli, c'est que l'envoi était du fait d'un robot (puisqu'un humain n'aurait pas vu le champ) (ou d'un malandrin malintentionné) et donc du spam.

Sauf que depuis est arrivée l'ère de la *Single Page Application* et de la page web qui ne fonctionne qu'avec Javascript activé. Donc, les robots de spam se sont modernisés et utilisent probablement des outils de pilotage de navigateur, type Selenium . La conséquence, c'est qu'ils ont la même «vision» de l'application que les utilisateurs réels, et donc que les champs cachés seront ignorés.

Mais alors, comment se prémunir de ce genre de spam?

Est-on obligés de faire appel à des services tiers, comme semble l'indiquer l'immense majorité des ressources sur la question?

3. Deux solutions simples et efficaces

Il y a deux techniques très efficaces que je m'étonne de les avoir vu que très rarement.

3.1. L'arme nucléaire : interdire les liens dans votre formulaire

Tous ces spams automatisés n'ont pour but que de vous faire cliquer sur des liens malveillants. Donc, s'il n'y a aucune raison légitime de mettre des liens dans votre formulaire, vous pouvez tout simplement…

> *i* Interdire les liens dans votre formulaire (et prévenir les utilisateurs de ce fait).

Une simple détection sur `http://` ou `https://` supprime 90% de ceux que je reçois; une version plus poussée qui détecte `{texte}.{texte}/{texte}` avec `{texte}` des bouts de textes sans espaces les détecte *tous*.

3.2. Plus subtil et un peu plus risqué

Une autre possibilité c'est de jouer sur le fait que les robots remplissent les formulaires *rapidement*. **Trop vite** en fait. Dans ceux que je reçois, c'est entre 300 et 1200 ms, avec une moyenne autour de 600 ms. Et donc…

> *i* Il suffit de mesurer le temps mis entre l'affichage du formulaire et son renvoi, les réponses trop rapides seront comptabilisées comme du spam.

Le plus simple, *sans avoir à gérer de session*, est d'envoyer une date (avec une précision à la seconde ou mieux) dans le formulaire. Le formulaire renvoie cette date sans la toucher, le serveur peut facilement calculer le temps écoulé.

> ! Vous **devez** utiliser une **horloge monotone** pour cette utilisation, et **pas** l'horloge du système.

Une horloge monotone, c'est une horloge qui avance toujours au même rythme, quoi qu'il se passe. À la différence de l'horloge système, qui est généralement à l'heure légale ou à UTC. Or, cette heure bouge de façon non-linéaire: changements de fuseaux horaires, ajustements manuels, auto-ajustement avec NTP … et donc tout calcul précis de durée basé sur ce type d'horloge peut être faux et mener à des résultats aberrants.

Par exemple en Java, il faut utiliser `System.nanoTime()` et pas `System.currentTimeMillis()` (ni une nouvelle date, elles se basent sur ce dernier appel).

4. Limites

Les principales limites de ces systèmes sont:

1. Aucune résistance à une attaque ciblée: l'attaquant qui a la patience de faire un script spécialement pour votre formulaire ne mettra pas longtemps à contourner les protections.
2. Ça peut quand même ennuyer l'utilisateur; soit s'il avait en fait une vraie raison de mettre un lien à laquelle vous n'avez pas pensée, soit si le temps de blocage du formulaire est trop court (pensez au cas où presque tout est pré-rempli par le navigateur et où l'utilisateur a prévu un copier/coller du texte principal).

Néanmoins en pratique ça couvre une bonne partie des cas d'usage, **sans le moindre inconvénient ergonomique pour l'utilisateur**.

Et c'est le principal.

Logo: une boite de spam, la viande en boite .