

NAXSI, un pare-feu applicatif pour NGINX

Matthieu Guionnet

LISBP

135, avenue de Rangueil

31 000 Toulouse

Résumé

Ce document présente le pare-feu applicatif web Naxsi. C'est un module optionnel au serveur http Nginx. Il fonctionne sur le principe de l'apprentissage par listes blanches. Les sources du programme sont téléchargeables depuis le site du projet : <https://github.com/nbs-system/naxsi>.

Mots clefs

Sécurité, Serveur web, pare-feu applicatif

1 Rapide état des lieux des solutions libres disponibles

Un pare-feu applicatif web a pour but de filtrer les requêtes http afin de détecter les tentatives de compromission d'une application web. ModSecurity (<http://www.modsecurity.org/>) a longtemps été le seul pare-feu applicatif web libre et crédible. Naxsi est un projet de l'Open Web Application Security Project (<http://www.owasp.org/>), organisation à but non lucratif. Naxsi et ModSecurity partagent donc les mêmes objectifs mais ils diffèrent sur les moyens pour y parvenir. Naxsi, pour Nginx Anti Xss & Sql Injection, est un module pour le serveur Nginx (<http://nginx.org>). Ce dernier est un serveur HTTP et proxy inversé mais aussi un proxy pour les protocoles de mail POP3 et IMAP. Nginx est actuellement le 3ème serveur le plus utilisé dans le monde pour héberger les pages web. Son succès est certainement aussi celui des serveurs d'application web tel Ruby on Rails qui ont mis en avant ses bonnes performances ainsi qu'une faible empreinte mémoire. Nginx est apparu en 2002 à peu près au même moment que d'autres projets de serveurs web libres tel que Lighttpd (<http://www.lighttpd.net/>) ou Cherokee (<http://cherokee-project.com/>). Ils ont en commun d'être des réponses au problème des dix mille clients « the C10K problem » (<http://www.kegel.com/c10k.html>). Ce problème a popularisé l'utilisation de boucles d'événements rapides disponibles sur les noyaux des systèmes d'exploitation modernes comme par exemple epoll sur Linux ou kqueue sur FreeBSD.

Bien que ne répondant pas au problème des dix mille clients, le serveur web développé il y a bien longtemps qui domine largement depuis plus d'une décennie le classement du nombre de pages web hébergées est Apache Httpd (<http://httpd.apache.org>). Sa position dominante peut s'expliquer par les habitudes des administrateurs mais aussi par la pléthore de modules optionnels disponibles dont le pare-feu applicatif Modsecurity.

Naxsi se démarque de Modsecurity par sa gestion des règles de filtrage. Modsecurity utilise une base de signatures pour détecter spécifiquement les requêtes malintentionnées. Sur le principe des logiciels antivirus, il faut ainsi se faire ou se doter d'une base de signature puis ensuite la maintenir. On peut disposer gratuitement de certaines bases ou en acquérir, comme celle de la société Trustwave qui propose 18000 règles spécifiques à intégrer dans Modsecurity. Ces signatures sont, pour la plupart, basées sur des expressions régulières. Les attaquants d'un site web, pour contrer ce type de contrôle complexifient leurs requêtes afin de les rendre évasives. En face, les défenseurs doivent répondre par une complexification des expressions régulières employées. Cette escalade n'est pas sans impact sur les ressources nécessaires afin de mettre en œuvre ce type de pare-feu.

Naxsi propose bien moins de règles. Par exemple, dans la version 0.53, il y a seulement 41 règles actives par défaut, il ne comporte pas d'expression régulière mais un système de type bayésien comme on en retrouve dans les anti-pourriels de nos systèmes de mail. Une note est attribuée par règle et par type d'attaque possible. Elle s'additionne à celles des autres règles qu'une requête pourrait rencontrer. Donc plus la note est élevée, plus la requête est potentiellement malintentionnée. On détermine un seuil par type d'attaque, au delà duquel on considère cette attaque avérée. Pour limiter les faux-positifs il

faut ajouter à ces règles une liste blanche que l'on peut créer à la suite d'une période d'apprentissage. Avec ces principes, Naxsi atteint ses objectifs qui sont de conserver les performances du serveur web Nginx et d'assurer une protection contre les injections et les XSS (cross-site scripting) sans nécessiter beaucoup de maintenance.

Débuté récemment, en 2011 par Thibault Koechlin, le développement de Naxsi est sponsorisé par son employeur, la société NBS System, spécialiste dans l'hébergement et la sécurité des sites web (<http://www.nbs-system.com>). Naxsi est placé sous la protection de la licence libre GNU GPLv2 (<http://www.gnu.org/licenses/gpl-2.0.html>) et à ce titre on dispose donc des sources permettant son utilisation après un éventuel audit.

2 Mise en œuvre

2.1 Installation

Les numéros de versions disponibles sont pour le moment très inférieurs à 1 afin de bien signifier que le développement est encore loin d'être achevé mais aussi que les fichiers de configurations, les paramètres et même les commandes peuvent être amenés à changer fortement d'une version à l'autre. Le site de développement projet a aussi récemment changé pour le site `github` (<https://github.com/nbs-system/naxsi>) et il faut regarder du côté des pages *wiki* de ce site pour trouver les documentations à jour relatives à l'installation et la configuration de Naxsi avec Nginx.

Des binaires sont disponibles dans les dépôts de certaines distributions Linux comme *Debian* et *Ubuntu* ainsi que dans ceux des systèmes *NetBSD* et *FreeBSD*.

Nginx ne propose pas de module dynamique comme Apache Httpd. L'installation depuis les sources, nécessite donc de compiler Nginx pour y intégrer le module Naxsi. Nginx est écrit principalement en langage C tout comme le cœur de Naxsi. Cette opération est rapide et aisée car ce sont deux projets de tailles relativement petites dans leurs catégories respectives et Nginx est maintenant un projet mature.

Le projet Naxsi comporte aussi une partie en langage *Python* pour la génération des listes blanches et de pages de rapport des détections faites par Naxsi, au format *html*. Cette dernière partie, une fois installée se résume à l'utilisation de la commande `nx_util.py` depuis un shell.

2.2 Configuration

Pour la configuration de Naxsi, la partie la plus ardue est peut-être de savoir configurer un serveur web Nginx. Une fois fait, les ajouts concernant le pare-feu Naxsi sont minimes, il convient de suivre les conseils donnés sur les pages *wiki* du projet (<https://github.com/nbs-system/naxsi/wiki>) pour cela.

On pourra résumer la configuration de Naxsi pour un site hébergé par Nginx ainsi :

- définir son mode de fonctionnement, apprentissage ou non ;
- indiquer quelles sont les règles à utiliser et leurs scores ;
- définir les seuils de détections ;
- que faire en cas de détection, typiquement quelle page d'erreur *HTTP* retourner ;
- insérer la liste blanche des requêtes *HTTP* obtenue après la période d'apprentissage.

Lorsque l'on a un serveur Nginx avec Naxsi correctement configuré, l'activité de notation des requêtes *HTTP* par Naxsi se retrouvera dans les journaux d'erreurs de Nginx. Il convient de passer par une période d'apprentissage afin d'établir, à l'aide de ces journaux et de la commande `nx_util.py`, une liste blanche. Cette liste correspond aux requêtes *HTTP* que l'on sait être des faux positifs pour Naxsi. La liste blanche permettra au site de fonctionner normalement en indiquant à Naxsi de ne plus évaluer ces requêtes.

Pour établir correctement les listes blanches, on conseille d'utiliser, sur un réseau interne sûr et contrôlé, un serveur Nginx avec Naxsi en position de relais vers le site à protéger. Ainsi les requêtes qui ont été détectées sur ce relai comme des tentatives de détournement du site seront assurément des faux-positif. La liste blanche ainsi générée sera par la suite ajoutée dans la configuration du serveur qui aura effectivement la tâche de protection du site vis à vis de l'extérieur (voir figure 1).

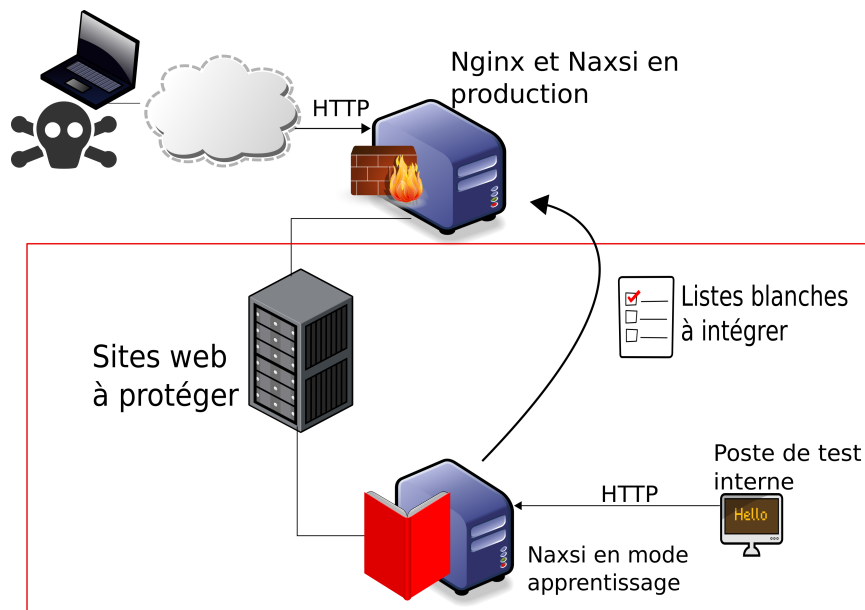


Figure 1 - Illustration de la configuration conseillée pour mettre en place la protection proposée par Nginx et Naxsi

La période d'apprentissage n'est pas négligeable. Il faudrait, pour bien faire, valider l'ensemble des formulaires présents sur le site web à protéger. L'attention devra être portée principalement sur l'apparition des caractères recherchés par Naxsi dans les valeurs passées par un formulaire.

Par exemple, dans une page d'authentification typique à un site web, il est généralement demandé, en retour, le login et le mot de passe et il sera nécessaire de se poser des questions comme « l'application accepte-t-elle la présence d'une quote «'» dans le login ? ».

Vous avez aussi la possibilité de modifier les seuils de détection des attaques proposés par défaut par le projet Naxsi. Cela aura un impact sur le nombre d'exception contenu dans les listes blanches mais cette manipulation est peu documenté.

2.3 Exemple de code failible

Naxsi se propose de nous prévenir principalement des injections et des XSS (Cross Site Scripting) dans une application web. Les injections concernent plus particulièrement le serveur web et les données qui y sont stockées. Une attaque de ce type réussie, permettra à son initiateur de récupérer des données non accessibles et/ou d'en détruire. Si vous souhaitez voir Naxsi à l'œuvre avec une application web mal écrite voici un extrait de code Python pour exemple :

```
def my_view(request):
    if request.GET:
        q = request.GET['q']
        results = commands.getoutput('sqlite3 -line badproject/db/database.db\
                                     \'select nom, identifiant from utilisateur where\
                                     nom = \'' + q + '\\'\')
    else:
        q = ''
        results = ''
    return {'results': results}
```

Le site attend une variable nommée *q* retournée par un formulaire par la méthode *HTTP get*. Cette variable est passée dans une commande *shell* qui interroge une base de donnée relationnelle selon une question bien définie « donne moi le nom et l'identifiant de la personne dont le nom est ».

Le plus gros défaut de cette fonction est que la vérification de la variable passée *q* est quasi nulle et il est donc possible d'entrer du code *SQL* supplémentaire dans la requête.

Si vous hébergez ce programme sur un serveur web, vous pourrez ainsi obtenir l'ensemble de la table utilisateur par l'adresse suivante :

```
http://monsite/?q=%3Bselect+*+from+utilisateur%3B
```

Ou bien choisir, par exemple, de détruire l'ensemble des entrées de cette table :

```
http://monsite/?q=%3Bdelete+from+utilisateur%3B
```

Ce programme est donc sujet à une faille de type injection de code SQL.

3 Retour d'expérience

Les objectifs de rapidité de Naxsi sont atteints. Avec l'exemple de code faillible donné en chapitre précédent et des outils de mesure de performances tel que `weighttp` (<http://redmine.lighttpd.net/projects/weighttp>), vous pourrez constater par vous même que l'impact négatif de ce programme sur les performances du serveur web Nginx sont aussi faibles qu'annoncé.

Les idées mises en œuvre dans ce projet de pare-feu applicatif web sont particulièrement intéressantes. Ce projet apporte une solution innovante dans son domaine mais il y a peut-être encore trois problèmes éventuels qui pourraient freiner son adoption :

- Nginx est encore assez peu connu des administrateurs systèmes comparé à Apache Httpd. On retrouve fréquemment l'utilisation des modules d'authentification CAS d'apache dans les structures concernés par les JRES. Mais Nginx et Naxsi pourraient tout de même être utilisés en proxy inversé devant le vénérable Apache comme illustré dans la figure 1.
- Lors d'un déploiement, l'efficacité d'une phase d'apprentissage est difficilement évaluable. De plus, un administrateur système pourrait avoir l'impression d'effectuer le travail que le développeur web n'a pas effectué en validant les variables de formulaires. Et il n'aura pas tort... Si, à l'inverse des petites structures, ce n'est pas la même personne, il vaudrait mieux avoir de bonnes relations entre les personnes ou les services.
- Le projet est encore jeune et les changements de version peuvent être très fréquents. Ainsi, récemment, suite à la sortie de la version 0.53 de Naxsi et des premiers retours qui ont suivi, le mainteneur du projet, Thibault Koechlin a rapidement déclassé cette version, la passant de *stable* à *brouillon* et ne sortant une version 0.53-1 qu'après une version intermédiaire, marquée en *release candidate*.

Les développeurs et notamment le mainteneur, s'expriment assez fréquemment sur son futur auprès de la communauté des utilisateurs. Le développement va s'orienter vers la mise en place de plus de tests qualités et d'audit de code comme annoncé dans un billet du blog sur le projet <http://blog.memze.ro/>. Les utilisateurs et contributeurs occasionnels obtiennent rapidement des réponses aux diverses questions ou propositions qu'ils pourraient formuler sur la liste de diffusion du projet. Pour terminer, sachez tout de même que la société NBS System qui sponsorise ce projet l'utilise pour protéger les plus de 2500 sites qu'elle héberge.