Un *reverse proxy*, pour quoi faire ?

Pascal Cabaud
UFR EILA, Université Paris Diderot

Résumé

Les systèmes d'information de nos établissements comportent de plus en plus d'applications Web. L'ensemble doit être rendu cohérent pour l'internaute. Ces applications peuvent être lourdes et nécessiter plusieurs machines, donc éventuellement une répartition de charge. En insérant un système intermédiaire entre l'internaute et les serveurs Web, il est possible d'agréger des ressources diverses en un site Web unique et de répartir la charge entre plusieurs serveurs.

Les firewalls installés il y a quelques années ne protègent plus guère ces serveurs Web : les filtres au niveau IP s'assurent bien que le port de destination est 80 ou 443, au mieux ils vérifient les options TCP mais en aucun cas ils n'inspectent le contenu des requêtes. Or, les applications Web sont vulnérables aux contenus malicieux (injections diverses de code ou de données) qui transitent via HTTP(S). Un filtrage au niveau applicatif s'avère donc aujourd'hui indispensable.

Nous disposons pour cela de nombreux logiciels open source tant pour router et répartir la charge entre des serveurs Web que pour authentifier les utilisateurs ou filtrer les requêtes malicieuses.

Mots clefs

Firewall applicatif Web, routage applicatif Web, répartition de charge, sécurité des applications Web, *reverse proxy*

1 Introduction

Nous avons désormais besoin d'héberger moult applications Web. Certains d'entre nous, après avoir mis des ressources à disposition sur le réseau (imprimantes *via* LPR, fichiers *via* NFS ou SMB, *etc*.), migrent tout ou partie de ces ressources vers des interfaces Web, tendance qui a commencé avec les webmails et qui va en s'accentuant avec la mode du Web 2.0. Dans le même temps, nos utilisateurs sont de plus en plus dépendants de ces applications. Or, des vulnérabilités dans ces mêmes applications sont publiées tous les jours, parfois sans qu'un correctif ne soit disponible. Cette problématique est suffisamment importante pour qu'une initiative libre consacrée à l'étude de la sécurité et la publication de nombreux guides autour des applications Web ait vue le jour : l'*Open Web Application Security Project* (*cf.* [1]) ; la vitalité du projet témoigne des enjeux.

Parmi ces applications Web, souvent très différentes, certaines imposent des pré-requis ou des contraintes antinomiques (par exemple des réglages globaux dans `php.ini`). De plus, un site Web se compose souvent de plusieurs « parties » : des pages statiques comme des contenus dynamiques divers (lesquels sont parfois lourds en CPU). Grâce à la virtualisation, ces applications s'installent facilement sur des systèmes distincts et ce, à moindre coût. Nous pouvons cacher à l'utilisateur la complexité induite par cette multiplicité et « unifier » l'ensemble, en offrant une vue cohérente du système d'information à travers un unique site Web (au prix d'un dernier effort sur les feuilles CSS).

Un autre avantage important à ce principe « un système pour une application » réside bien évidemment dans les possibilités d'améliorations des mesures de sécurité que cela confère (pas seulement au niveau application mais aussi aux niveaux réseau ou système).

2 Le *reverse proxy*

La pièce maîtresse du dispositif est le *reverse proxy* ou mandataire inverse. Ce système reçoit le trafic HTTP et HTTPS venant de l'extérieur. Il le traite non pas comme un serveur Web classique en répondant par des données stockées sur disque ou générées dynamiquement mais en transmettant la requête à un autre serveur et en renvoyant la réponse de ce dernier au client. Nous insérons donc un système entre l'internaute et le serveur *Web* réel.

L'installation d'un *reverse-proxy* permet de mettre en place de tout ou partie des mesures de protection suivantes :

—couper tout accès direct depuis Internet vers les serveurs d'applications Web (adressés en RFC 1918 par exemple) et inversement,

—couper tout accès direct entre le bastion, connecté à Internet, et les serveurs de données DAV, LDAP et SQL,

—surveiller et auditer le trafic après déchiffrement du trafic HTTPS.

Suivant les fonctionnalités offertes par l'implémentation, le mandataire peut aussi :

—prévenir les dénis de service et autres attaques de mots de passe par force brute,

—détecter les requêtes malicieuses voire les bloquer (détection et prévention d'intrusions),

—assurer l'authentification des accès à certaines parties d'un site,

—répartir la charge entre plusieurs systèmes,

—mettre en cache les fichiers statiques.

Il faut voir ce système comme un routeur — éventuellement filtrant —, ce qu'il est, non pas au niveau 3 mais au niveau 7 — tout comme une passerelle SMTP par exemple. Ce système est donc critique et gagne à bénéficier d'un système de haute disponibilité.

Le choix logiciel est vaste : outre Apache[1] et son module `mod_proxy.so`, citons aussi par les *daemons* applicatifs les plus utilisés : lighttpd et ngnix (deux serveurs Web), Pound (*reverse-proxy* et répartiteur de charge), Squid (un *proxy* classique, aussi utilisable comme *reverse-proxy*) et Varnish (un accélérateur HTTP).

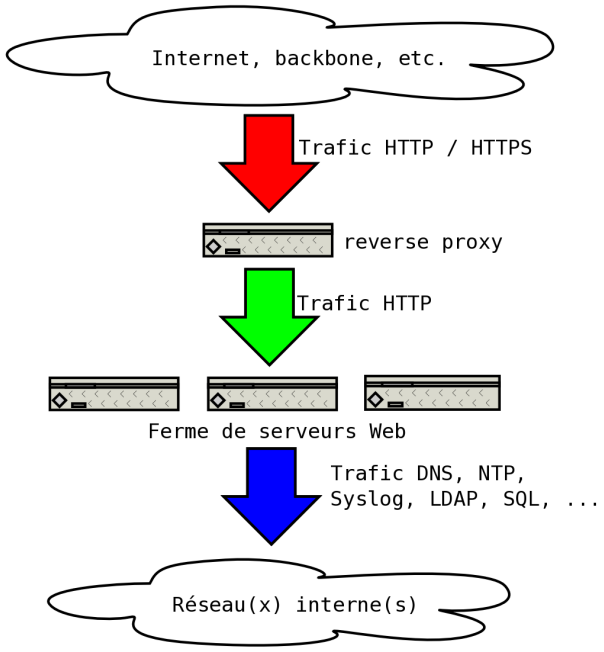

Figure 1 - *Cheminement des données*

Dans l'exemple suivant, un relai Apache renvoie `/foo` à un premier serveur, `/bar` à un second. Pour l'internaute, les URL visibles seront uniquement `http://www.u-exemple.fr/foo` et `http://www.u-exemple.fr/bar`. Ce mandataire peut ainsi faciliter l'agrégation de différentes ressources hébergées sur plusieurs systèmes tout en masquant ces derniers à l'internaute. Avec Apache et `mod_proxy.so`, cela donne :

```
ProxyRequests off
ProxyErrorOverride on
<Proxy *>
  Allow from all
</Proxy>
ProxyPass        /foo http://server_a/foo
ProxyPassReverse /foo http://server_a/foo
ProxyPass        /bar http://server_b/bar
```

[1]Avec Apache 1.3, nous avons rencontré par le passé des problèmes récurrents au delà de 16 *virtual hosts* en HTTPS, utiliser plutôt une version 2.0 ou 2.2.

```
ProxyPassReverse /bar http://server_b/bar
```

La méthode HTTP `TRACE` permet de suivre le cheminement d'une requête de serveur en serveur à la manière de `traceroute(8)`. On peut le voir, par exemple en utilisant `telnet(1)` :

```
$ telnet www.u-exemple.fr 80
TRACE / HTTP/1.0
Host: www.u-exemple.fr
[...]
TRACE / HTTP/1.0
Host: www1.priv.u-exemple.fr
[...]
```

Figure 2 - *Test de la méthode `TRACE`*

On voit ici que `www.u-exemple.fr` renvoie les requêtes à `www1.priv.u-exemple.fr`. Si on veut masquer cette architecture, il faut désactiver la méthode `TRACE` ; avec Apache, on peut soit régler `TraceEnable` à `Off`, soit utiliser le module `mod_rewrite.so` avec le drapeau `F` (*forbidden*) :

```
RewriteEngine on
RewriteCond %{REQUEST_METHOD} ^TRACE
RewriteRule .* - [F]
```

Les autres méthodes `CONNECT`, `DELETE`, `OPTIONS` et `PUT` peuvent se bloquer de la même manière. Ce même module `mod_rewrite.so` est le couteau suisse de l'administrateur Web et permet, *via* des règles, la manipulation d'URL. Ces mêmes règles peuvent, avec le drapeau `P` (*proxy*), appeler `mod_proxy.so`.

La mise en place des applications derrière un *proxy* est parfois difficile du fait des liens hypertextes. Plutôt que de renvoyer `http://www.u-exemple.fr/appli/myapp` vers `http://appserver_1/myapp`, on aura tout intérêt à reproduire sur le serveur interne `appserver_1` l'arborescence telle que vue par l'internaute *i. e.* renvoyer vers `http://appserver_1/appli/myapp` et y disposer les fichiers dans `${DOCUMENT_ROOT}/appli/myapp`, la gestion des liens hypertextes s'en trouvera alors grandement facilitée : il ne reste plus alors que le nom de l'hôte et éventuellement le protocole à modifier s'ils sont mentionnés. Les autres alternatives sont :

—modifier l'application (pas toujours faisable ni souhaitable),

—jouer avec des directives comme : `Alias /myapp /path/to/htdocs/myapp`,

—ruser avec des règles `mod_rewrite.so`,

—réécrire à la volée les liens avec le filtre `mod_proxy_html.so`.

Il faut noter qu'au fil du temps, de plus en plus d'applications Web sont prévues pour pouvoir passer au travers d'un serveur mandataire. Cela passe alors souvent par des variables à configurer comme par exemple pour DokuWiki :

```
$conf['baseurl'] = "http://www.u-exemple.fr";
```

3 Authentification et chiffrement

Le mandataire a besoin d'une adresse IP par *virtual host* HTTPS[2]. Ces adresses et les noms correspondants seront visibles de l'extérieur ; ce seront même les seuls visibles de l'extérieur. Le mécanisme d'hôtes virtuels rend possible la centralisation de plusieurs sites (tous ?) sur un même mandataire.

Outre la confidentialité des données et la vérification que le serveur est le bon, la couche SSL/TLS offre la possibilité d'authentifier les utilisateurs avec un certificat et ce, sans interroger une base de mots de passe (LDAP, SQL, fichiers

[2]Il existe des parades :

- les certificats « étoile » (`*.u-exemple.fr`)
- les certificats renseignants plusieurs noms FQDN (ajouter des attributs `subjectAltName` lors de la création de la demande de certificat ; cet attribut prend des arguments préfixés par `DNS:` (ou `IP:`) et séparés par des virgules).

Enfin, dans les années à venir, *Service Name Indication* (RFC 3546) devrait se développer. Plutôt que d'utiliser SSL, la connexion HTTP est chiffrée par TLS ; lors de la requête, l'entête `Host:` étant connu, le serveur présente le certificat du *virtual host* demandé. Pour l'heure, cette extension n'est pas encore disponible sur tous les logiciels, clients comme serveurs.

`htpasswd(5)` et `passwd(5)`…). Nous disposons ainsi d'une authentification forte sans que des mots de passe ne transitent ni ne soient stockés sur une machine directement accessible depuis Internet. Par exemple, avec Apache et `mod_ssl.so`, la configuration contiendra :

```
SSLVerifyClient none
SSLCACertificateFile /etc/httpd/ssl.crt/ca-root.crt
[...]
<Location /secured>
  SSLVerifyClient require
  SSLVerifyDepth 1
</Location>
```

Ici, un client n'accède à `https://www.u-exemple.fr/secured/` qu'avec un certificat signé par l'autorité[3] dont le certificat racine est stocké dans `ca-root.crt`. Outre `mod_ssl.so`, il existe moult autres modules, à commencer par `mod_gnutls.so`, qui offre des fonctionnalités similaires, mais aussi de quoi s'authentifier *via* Kerberos, PAM, SASL…

Après traitement par le mandataire, le trafic HTTPS peut être déchiffré : il peut donc, lui aussi, être audité sans difficulté, par exemple avec Snort, sur la patte interne du *proxy*[4].

Enfin, on ajoute relativement facilement une couche d'authentification à une application avec le fournisseur de services de Shibboleth. Il ne reste plus qu'à modifier l'application pour utiliser les variables d'environnement renvoyées `HTTP_SHIB_*` par le fournisseur d'identité.

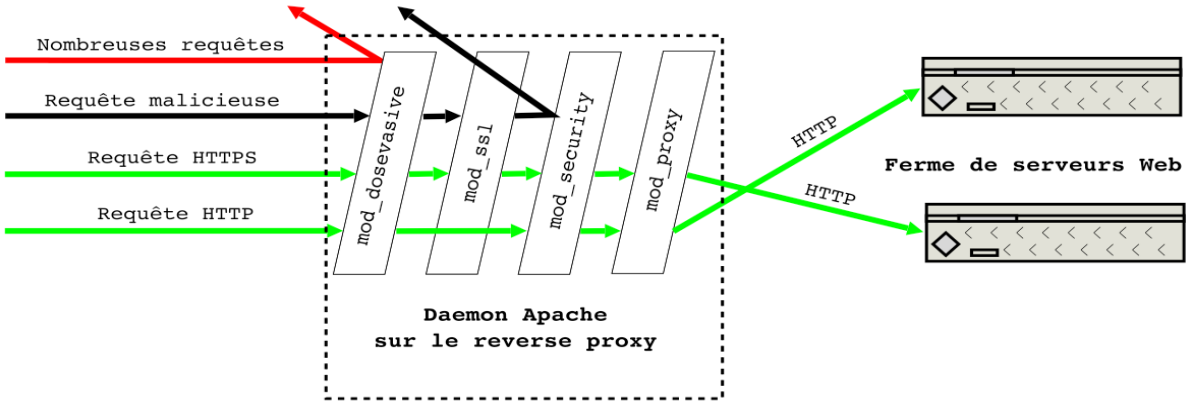

Figure 3 - *Traitements par Apache*

4 Détection et prévention d'intrusions : *Web Application Firewall*

Nous pouvons nous protéger des requêtes malicieuses (injections SQL, exécution de commandes arbitraires, *cross site scripting*, *etc.*) vers nos applications Web avec `mod_security.so` (*cf.* [8]).

Ce module est publié sous GPL par *Breach Security Inc*. Il s'utilise soit en mode prévention d'intrusions (bloquant) soit en mode détection (journalisation uniquement). Il est vivement recommandé de commencer par utiliser le mode détection sans quoi de nombreuses applications seront cassées. Le module utilise des règles et permet ainsi de limiter la taille des requêtes `POST`, de vérifier le contenu de ces requêtes et des réponses (par expressions régulières), d'en bloquer certaines, de passer tout fichier *uploadé* à un exécutable (par exemple pour l'analyser avec un antivirus)… Le module permet d'analyser le trafic HTTP aux niveaux 6 et 7 du modèle OSI : les commandes HTTP mais aussi les données sont passés au crible.

Ce module se base sur des règles ; ces dernières sont applicables aux différentes phases (numérotées de 1 à 5) du traitement par Apache : `REQUEST_HEADERS` puis `REQUEST_BODY`, `RESPONSE_HEADERS` puis `RESPONSE_BODY` et enfin `LOGGING`. Ces règles ont différentes actions (à commencer par `deny`, `log` et `pass`).

Il était fournit avec un jeu de règles génériques (*Core Rules Set)* qui est désormais indépendant (*cf.* [11]). Un autre jeu de règles, éditées par le site *GotRoot* (*cf.* [12]), peut les compléter. Dans les deux cas, des scripts sont fournis pour mettre à jour

[3]Pour utiliser des sous-autorités, augmenter le paramètre `SSLVerifyDepth`.

[4]Parfois, Snort n'analyse que le trafic entrant ; il faut alors ruser en créant une interface TAP et un pont avec la patte physique interne pour faire écouter cette interface virtuelle à Snort.

ces jeux de règles automatiquement (sic !). L'administrateur peut évidemment écrire ses propres règles. Il peut s'aider pour cela de ModProfiler (*cf.* [13]) ou d'autres outils libres comme WebScarab, Paros ou Burp (*cf.* [14], [15] et [16] respectivement).

`Mod_security.so` génère des journaux conséquents. Pour leur analyse, *Breach Security Inc.* fournit une console (actuellement disponible gratuitement pour la gestion d'au plus trois serveurs) ; sinon, le développement d'outils spécifiques s'impose étant donné le format de ces journaux. Enfin, `mod_security.so` envoie aussi les alertes *via* `syslog(3)`.

L'installation de `mod_security.so` a un coût d'entrée (en temps) mais le gain justifie son utilisation. Concernant le coût, outre le temps de l'administrateur, le module est peu gourmand en mémoire ; en revanche, les temps de réponse peuvent être pénalisés : suivant le jeu de règles chargé, nous observons un coût pouvant atteindre 45 %.

En cas de blocage et dans l'urgence, on trouve aisément le numéro de la règle bloquante (dans les journaux, chercher `id: 12345`) et la directive suivante la désactive localement :

```
<Location /path/to/blocked/app.php>
  SecRuleRemoveById 12345
</Location>
```

Pour autant, disposer d'un filtre applicatif ne dédouane pas de se tenir informé des vulnérabilités du moment ni d'appliquer les mises à jour. Comme toujours, il convient de rester circonspect quant aux résultats pour ne pas avoir un faux sentiment de sécurité.

Sur les plateformes Microsoft, mentionnons WebKnight (*cf.* [17]), autre *Web Application Firewall* libre, qui fonctionne avec les serveurs compatibles ISAPI (Apache et IIS). Sur les serveurs hébergeant du code PHP, on peut protéger ce dernier par le patch et le module Suhosin (*cf.* [18]) ; le code peut quant à lui être protégé avec Munin IDS ou PHP IDS (*cf.* [19] et [20]), deux bibliothèques PHP.

5 Dénis de service, bande passante, force brute

Ici aussi les outils sont nombreux à commencer par les mécanismes de gestion de la bande passante. La logithèque de modules Apache est apparemment riche ; elle compte notamment `mod_evasive.so` (aussi connu sous le nom de `mod_dosevasive.so`). Si la lutte contre les dénis de service à proprement parler ne semble pas indispensable dans un environnement Éducation-Recherche, les attaques par force brute sont néanmoins monnaie courante et ces modules[5] pourraient aider à s'en prémunir. On trouve aussi `mod_qos.so` (*cf.* [21]) dont la directive `QS_SrvMaxConnClose` est efficace. Ceci étant, tous ces outils prennent difficilement en charge les clients utilisant des *proxies*.

`mod_qos.so` présente d'autres possibilités comme par exemple limiter pour une URL précise (un script CGI trop lent) le nombre de connexions ; il propose une liste blanche de clients « VIP » (réseau interne, *proxy*…).

Au printemps 2009, un papier est sorti sur les dénis de service HTTP par envoi de requêtes légitimes mais incomplètes ; avec l'article, un utilitaire est fourni avec toute la documentation utile (Slowloris, *cf.* [22]). Si le serveur est vulnérable sans qu'il soit possible ou souhaitable d'en changer, insérer un (autre ?) mandataire en amont comme nginx ou Varnish. Lors de la rédaction de ces lignes, il existe deux modules pour protéger Apache : `mod_antiloris.so` et `mod_noloris.so`. Le premier semble plus adapté aux serveurs utilisant plusieurs processus tandis que le second paraît plus efficace sur les serveurs multi-*threadés*.

Enfin, ne pas négliger la compression GZIP des données de type `text/*`. Cette compression peut se faire directement sur les serveurs ou sur le mandataire. Avec Apache 2.x, `mod_deflate.so` est fourni nativement. Le gain est difficilement mesurable efficacement mais le « ressenti utilisateur » s'en trouve amélioré sur des réseaux encombrés ou à faible bande passante.

6 Répartition de charge, performances, cache…

HAproxy (*cf.* [23]) est très souvent cité dans le contexte de la répartition de charge, de même qu'nginx et Varnish. Si on veut utiliser Apache, `mod_proxy_balancer.so` complète `mod_proxy.so` et transforme le mandataire en répartiteur de charge pour le protocole HTTP (AJP13 et FTP sont aussi disponibles).

[5] Des règles `mod_security.so` sont inutiles, elles ne s'appliquent qu'une fois la requête envoyée ; le principe de Slowloris consiste justement à ne jamais envoyer la séquence `\r\n`.

```
<Proxy balancer://app>
  BalancerMember http://vm-app-01:80/
  BalancerMember http://vm-app-02:80/
  BalancerMember http://vm-app-03:80/
</Proxy>
ProxyPass /appli balancer://app/
```

Quel que soit le répartiteur, la question se pose de l'algorithme de distribution. Le DNS offre un mécanisme basique de tourniquet (*round robin*). Le module Apache offre quant à lui trois possibilités :

—en répartissant les requêtes,

—en répartissant le volume de données traitées,

—en se basant sur les requêtes en attente.

Dans les deux premiers cas, on peut affecter un poids aux différents serveurs. Pour mettre en œuvre un suivi de session, utiliser un *cookie* de session (pour les applications intégrant une authentification interne par exemple) ; sur les serveurs, ajouter les *cookies* :

```
RewriteRule .* - [CO=BALANCEID:balancer.app_XX:.u-exemple.fr]
```

(faire varier XX) puis sur le *reverse proxy* activer le suivi :

```
<Proxy balancer://app>
  BalancerMember http://vm-app-01:80/ route=app_01
  BalancerMember http://vm-app-02:80/ route=app_02
  BalancerMember http://vm-app-03:80/ route=app_03
</Proxy>
ProxyPass /appli balancer://app/ stickysession=BALANCEID lbmethod=byrequests
```

Pour servir moult petits fichiers (images, feuilles de styles, scripts, …), Apache n'a pas bonne presse. Un serveur mandataire peut utilement faire office de cache. On peut aussi opter pour un serveur de contenu *i. e.* un serveur dédié aux fichiers statiques. Il existe une norme pour mettre en place ce type de service : *Edge Side Includes* ; la documentation de Varnish mentionne un support partiel. Si on veut utiliser Apache, on peut utiliser `mod_proxy_html.so` pour réécrire les URL voulues.

Enfin, Apache offre plusieurs types de cache : cache disque, cache mémoire et enfin cache de fichiers locaux. Le cache mémoire améliore évidemment les temps de transfert ; c'est le seul que nous utilisons, le gain observé est de l'ordre de 10 %.

7 Conclusion

Hiver 2009, nous avons eu à déplorer un incident sur une application Web. `Mod_security.so` n'était alors qu'en mode « audit » mais nous a alerté. Plusieurs fichiers ont été chargés sur un serveur *via* une injection de code dans une application PHP. Pourtant, l'attaquant n'a pas été en mesure d'en tirer parti : d'une part, les connexions SMTP ne passaient pas (script de *mass mailing*) ; d'autre part, ses scripts utilisant `wget(1)` ne trouvaient pas notre *proxy* (classique celui-là !) pour sortir.

La mise en place d'un mandataire inverse est relativement simple. Elle introduit une grande souplesse dans le traitement des requêtes au prix d'une faiblesse supplémentaire qui peut être prévue et maîtrisée. Devant la multiplication des applications incompatibles entre-elles, nous avons du les répartir sur des systèmes distincts ; la configuration s'est alors retrouvée éclatée sur ces systèmes. La mise en place d'un mandataire inverse a permis de centraliser à nouveau une partie de cette configuration. De plus, les journaux `access_log` ont eux aussi été centralisés.

Bibliographie

[1] `http://www.owasp.org`

[2] `http://modules.apache.org`

[3] `http://httpd.apache.org`

[4] `http://nginx.net`

[5] `http://www.lighttpd.net`

[6] `http://www.squid-cache.org`

[7] http://varnish.projects.linpro.no
[8] http://modsecurity.org
[9] http://www.jwall.org/web/policy/editor.jsp
[10] http://remo.netnea.com
[11] http://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project
[12] http://modsecurity.org/projects/modprofiler/
[13] http://downloads.prometheus-group.com/delayed/rules/
[14] http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
[15] http://www.parosproxy.org
[16] http://portswigger.net/proxy/
[17] http://www.aqtronix.com
[18] http://www.hardened-php.net/suhosin/
[19] http://code.google.com/p/muninids/
[20] http://php-ids.org
[21] http://mod-qos.sourceforge.net
[22] http://ha.ckers.org/blog/20090617/slowloris-http-dos/
[23] http://haproxy.1wt.eu