Un reverse proxy, pour quoi faire ?

Pascal Cabaud
UFR EILA, Université Paris Diderot

Position du problème

En 2009, tout passe par le Web ou presque :

- Mail
- Fichiers
- Calendriers
- Contacts…

Donc ces applications sont critiques.

Plan

1. Le *reverse-proxy*
2. Chiffrement et authentification
3. Filtrage des requêtes
4. Répartition de charge, performances

Le *reverse-proxy* (1/8)

Hébergement de sites et applications *Web* parfois incompatibles (par exemple `php.ini`)

➜ La virtualisation peut résoudre ce problème :

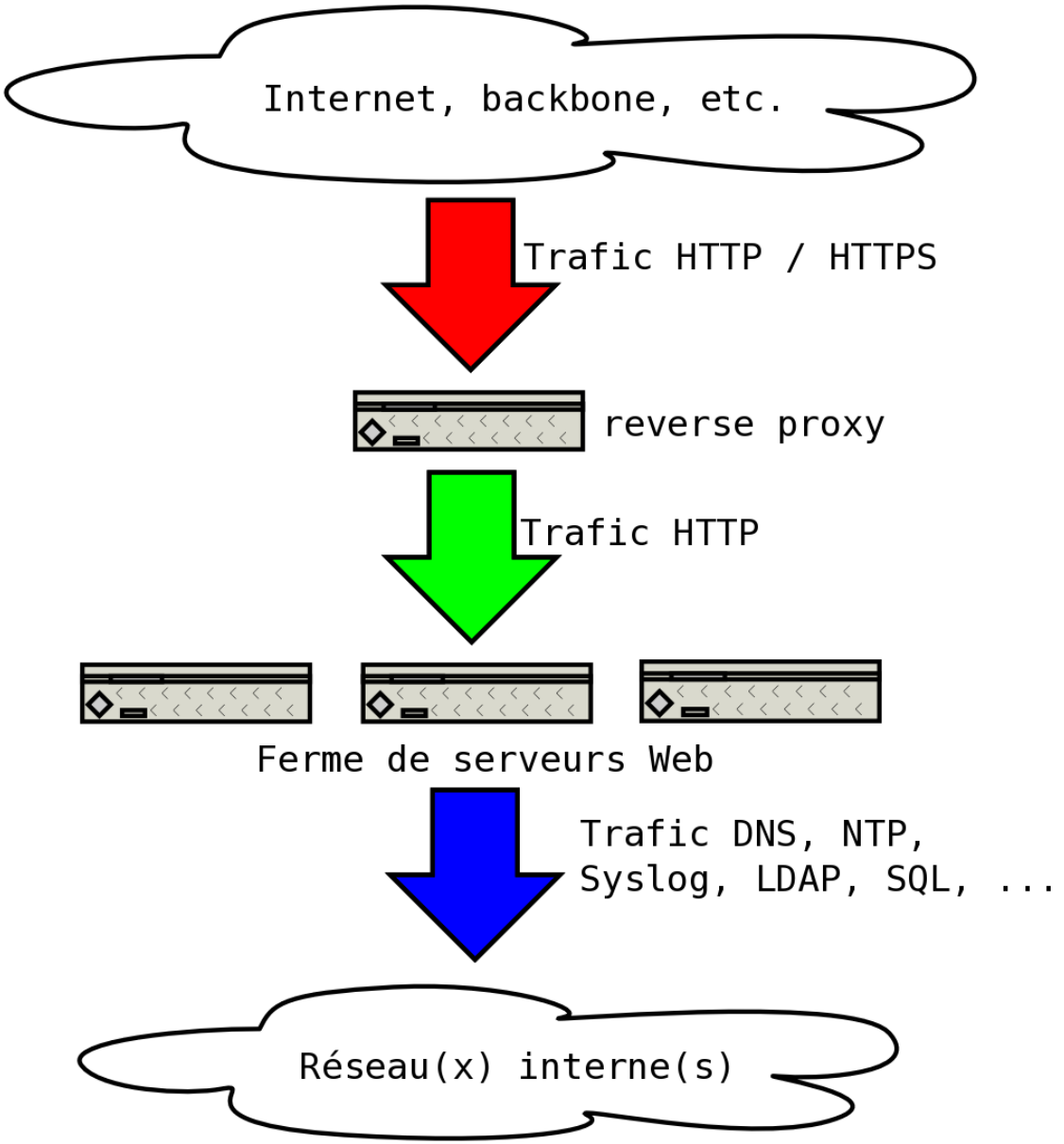
Applications compatibles sur un même système

Plusieurs systèmes :

1. morcellement
2. complexité

➜ Besoin de centraliser

Le *reverse-proxy* (2/8)



Le *reverse-proxy* (3/8)

Intermédiaire entre l'internaute et le serveur réel.

Permet :

- d'agréger diverses « parties » d'un site en un ensemble unique et cohérent
- de centraliser une partie de la configuration
- de couper les accès directs entre l'internaute et le serveur réel
- de surveiller le trafic HTTP et HTTPS

Le *reverse-proxy* (4/8)

Permet aussi de :

- détecter / bloquer les requêtes malicieuses
- répartir la charge
- protéger des dénis de service
- protéger contre les attaques par force brute
- mettre en cache certains objets

Le *reverse-proxy* (5/8)

Outils :

- Apache et `mod_proxy.so`
- Lighttpd
- nginx
- Pound
- Squid
- Varnish

Le *reverse-proxy* (6/8)

Exemple avec Apache : agrégation des ressources `http://www/foo` et `http://www/bar` :

```
ProxyRequests off
ProxyErrorOverride on
<Proxy *>
  Allow from all
</Proxy>

ProxyPass         /foo http://srv_a/foo
ProxyPassReverse /foo http://srv_a/foo

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

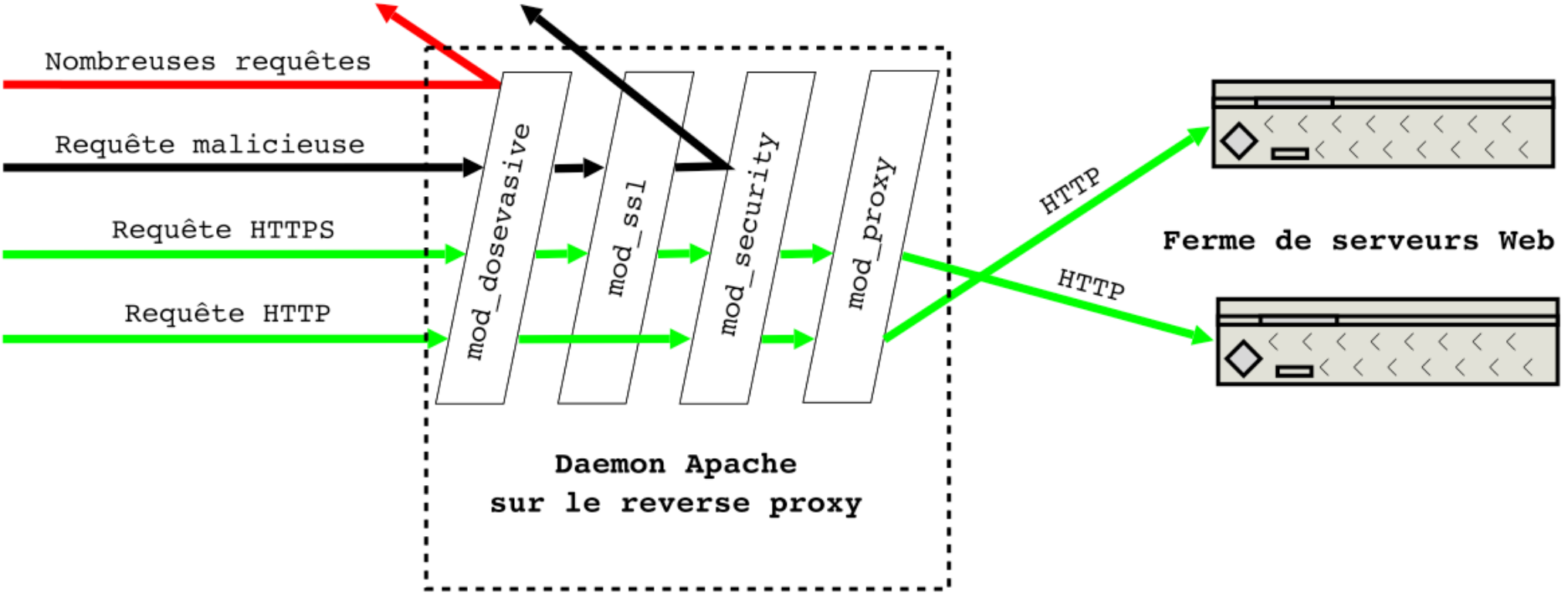
L'internaute ne voit que `www` et ignore tout de `srv_a` et `srv_b`.

Le *reverse-proxy* (7/8)

De `http://www/foo` à `http://srv_a/foo` : attention aux URLs « en dur ».

- reproduire l'arborescence sur `srv_a`
- modifier l'application
- `Alias /foo /path/to/foo`
- règles `mod_rewrite.so`
- les réécrire à la volée sur le proxy avec `mod_proxy_html.so`

Le *reverse-proxy* (8/8)



Chiffrement et authentification (1/3)

Pour chaque *vhost* HTTPS, il faut une IP (sauf certificats « étoile » : `*.u-exemple.fr` ou attribut `subjectAltName`).

Le *proxy* chiffre et déchiffre le trafic HTTPS.

Entre le proxy et les machines internes, le trafic transite en clair : audit facile.

Chiffrement et authentification (2/3)

Couche SSL/TLS permet authentification forte (par certificats) sans accès à une base quelconque (LDAP, etc).

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

Chiffrement et authentification (3/3)

Outils : par exemple Apache et les modules adaptés à PAM, OpenPGP, SASL, SSL/TLS…

En particulier, le fournisseur de services Shibboleth ajoute facilement une couche d'authentification. L'application doit alors utiliser les variables `HTTP_SHIB_XXX`.

Filtrage des requêtes (1/5)

Manipulations d'URLs par expressions régulières. Par exemple avec Apache et `mod_rewrite.so`, n'autoriser que ce qui est connu et interdire le reste :

```
RewriteRule /cgi-bin/foo.cgi / [P]
RewriteRule /cgi-bin/bar.cgi / [P]
RewriteRule /cgi-bin/.*\..*  - [F]
...
```

On peut aussi jouer avec les paramètres GET, les variables d'environnement (User-Agent), l'adresse IP du client...

Filtrage avancé (2/5)

Outils : Apache et `mod_security.so` :

`http://www.modsecurity.org`

Permet de se prémunir des requêtes malicieuses.

Basé sur des règles :

- à écrire soi-même,
- génériques, fournies par *Open Web Application Security Project* ou *AtomicCorp*

Filtrage avancé (3/5)

`mod_security.so` filtre, en entrée comme en sortie, en-têtes et corps, au cours des différents « stages ».

Possible après déchiffrement du trafic HTTPS.

Deux modes :

- **détection** (IDS) : journalisation uniquement
- **prévention** (IPS) : blocage des requêtes (ou des réponses)

Filtrage avancé (4/5)

Quelques points à régler dès l'installation :

- `SecRuleEngine DetectionOnly` pour commencer en mode détection seulement
- `SecRequestBodyInMemoryLimit` (max des `upload_max_filesize` des fichiers de configuration `php.ini` des serveurs proxyfiés)
- `SecResponseBodyLimit` (taille du plus grand document texte)
- Surveiller l'écart de performances

Filtrage : règles (5/5)

Règles génériques :

- basées sur des expressions régulières,
- protègent des vulnérabilités connues ou prévisibles,
- sont généralement utilisables en l'état.

En cas de faux positif, chercher `id:12345` dans les journaux et désactiver :

```
<Location /plop.php>
    SecRuleRemoveById 12345
</Location>
```

Répartition de charge, performances (1/5)

Répartition de charge :

- Apache avec `mod_proxy_balancer.so`,
- Haproxy,
- nginx,
- Varnish.

Répartition de charge, performances (2/5)

Apache avec `mod_proxy_balancer.so` :

```
<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/
```

Possibilité d'un suivi de session (avec un *cookie*) et de choisir l'algorithme de distribution ; *fail-over*.

Répartition de charge, performances (3/5)

Dénis de service

Apache dispose de plusieurs modules (`mod_dosevasive.so`, `mod_qos.so`) ; attention aux *proxies*.

`mod_qos.so` permet de limiter les accès à une application lente :

```
QS_LocRequestLimit /appli/foo.cgi 10
```

Ne pas oublier les outils classiques comme *fail2ban*.

Répartition de charge, performances (4/5)

Compression GZip

daemons classiques (avec Apache, `mod_gzip.so` (1.3), `mod_deflate.so` (2.x)) supportent la compression des contenus de type `text/*`.

Utile surtout pour les clients sur des réseaux à grande latence, faible bande passante.

Répartition de charge, performances (5/5)

Cache

Les *daemons* classiques supportent la mise en cache (disque et mémoire).

Pour servir moult petits fichiers, éviter Apache (pour nginx par exemple) ou mettre en cache (avec Apache, `mod_cache.so`).

Conclusion

Centralisation des accès, des configurations, des journaux. Homogénéisation pour l'internaute.

Possibilité d'économiser les adresses IP, de NATer, de filtrer, d'auditer... Détection et prévention d'intrusion.

Répartition des applications sur plusieurs systèmes. Éventuellement répartition de la charge d'une même application.

Questions ?

Merci !

Pascal Cabaud
http://www.eila.univ-paris-diderot.fr/~pc