Claire Billaud - 3ème année IS

MINI-PROJET : ETUDE D'UN MECANISME DE REDIRECTION DE PAGES WEB POUR AUTHENTIFIER UN UTILISATEUR WIFI

Principe :

On veut faire en sorte que le réseau interne de l'ENSEA devienne accessible par le WiFi grâce à une borne de connexion dans l'école. Le réseau sans fil étant réputé pour ne pas être du tout sécurisé, il est prévu de mettre en oeuvre un système de redirection permettant de demander à l'utilisateur un login et un mot de passe avant de l'autoriser à entrer sur le réseau.

Travail à faire :

Le serveur d'authentification doit pouvoir effectuer les tâches suivantes :

- Rediriger les requêtes de l'utilisateur WiFi en direction du réseau ENSEA vers le serveur
- Demander login/password au nouveau connecté
- Vérifier que ses données correspondent à des données existantes
- Si tout est bon, laisser passer l'utilisateur vers le réseau ; le serveur doit alors devenir "transparent" jusqu'à la déconnexion de l'utilisateur.

La gestion des login/password :

Le plus simple serait d'utiliser une base de données MySQL :

- simple d'utilisation, facile à programmer
- possibilité de stocker un grand nombre de logins différents
- facile à interroger avec une page PHP

La base de données contiendrait les logins et mots de passe des différents utilisateurs autorisés à accéder à l'Intranet et une page PHP très simple permettrait de vérifier la validité du login et du mot de passe entrés par l'utilisateur. Il serait même possible de garder une trace de toutes les tentatives de connexion effectuées à partir du WiFi.

L'inconvénient majeur est lié à la sécurité de la base de données. Il vaut mieux qu'elle ne soit pas installée sur le serveur (qui est le plus vulnérable de par sa position) car si le serveur est piraté, la base de données avec les mots de passe deviendrait accessible à n'importe qui. De plus, les mots de passe, à moins d'être changés régulièrement par les utilisateurs, sont fixes et peuvent donc plus facilement être interceptés. On pourrait cependant limiter ce risque en utilisant un protocole sécurisé (HTTPS) pour l'envoi des logins et mots de passe.

Cette solution nécessite l'installation sur la machine-serveur d'un serveur Web (Apache par exemple), d'un serveur de PHP, de MySQL et que la machine ait suffisamment de mémoire pour stocker la base de données (donc pas forcément beaucoup). D'une manière générale, le serveur n'a pas besoin d'être une machine très puissante (l'essentiel est que la connexion dispose d'une vitesse optimale et d'une bonne bande passante).

Après la connexion :

Une fois la connexion et la validation login/password résolues, le serveur doit devenir "transparent" pour l'utilisateur (donc ne pas redemander une authentification à chaque requête effectuée par l'utilisateur !)

Un point important : la sécurité du serveur

Le serveur est en effet en première ligne : il est donc très vulnérable. On peut le protéger des attaques extérieures en utilisant *iptables* du noyau Linux (voir plus loin) pour faire du routage à filtrage de paquets directement sur le serveur. On peut alors :

- **filtrer les paquets falsifiés** (paquets issus de pirates, qui viennent de l'extérieur en prétendant venir d'une adresse interne)
- **bloquer les protocoles dangereux ou inutiles** (en bloquant les paquets destinés aux ports dédiés à ces protocoles)

Mais aussi :

- bloquer les ports génériques non utilisés ou qui présentent un risque d'utilisation malveillante

Plus généralement, du point de vue de la sécurité, la règle la meilleure est celle du *"tout ce qui n'est pas explicitement autorisé est interdit"* : la règle de filtrage par défaut est de bloquer tous les paquets, puis d'autoriser uniquement ceux dont on veut permettre l'utilisation sur le réseau. On s'assure ainsi de n'accepter sur le réseau que les connexions qui ont été préalablement vérifiées grâce au système d'authentification.

La redirection :

La redirection peut se faire facilement en utilisant *iptables*. Cette fonctionnalité implantée dans le noyau Unix 2.4 permet au serveur de filtrer les paquets entrants et sortants, ainsi que les paquets passant par le serveur, et de les bloquer, de les laisser passer ou de les rediriger, en fonction de certains paramètres comme l'adresse IP de la machine qui les envoie ou le port auquel ils sont destinés.

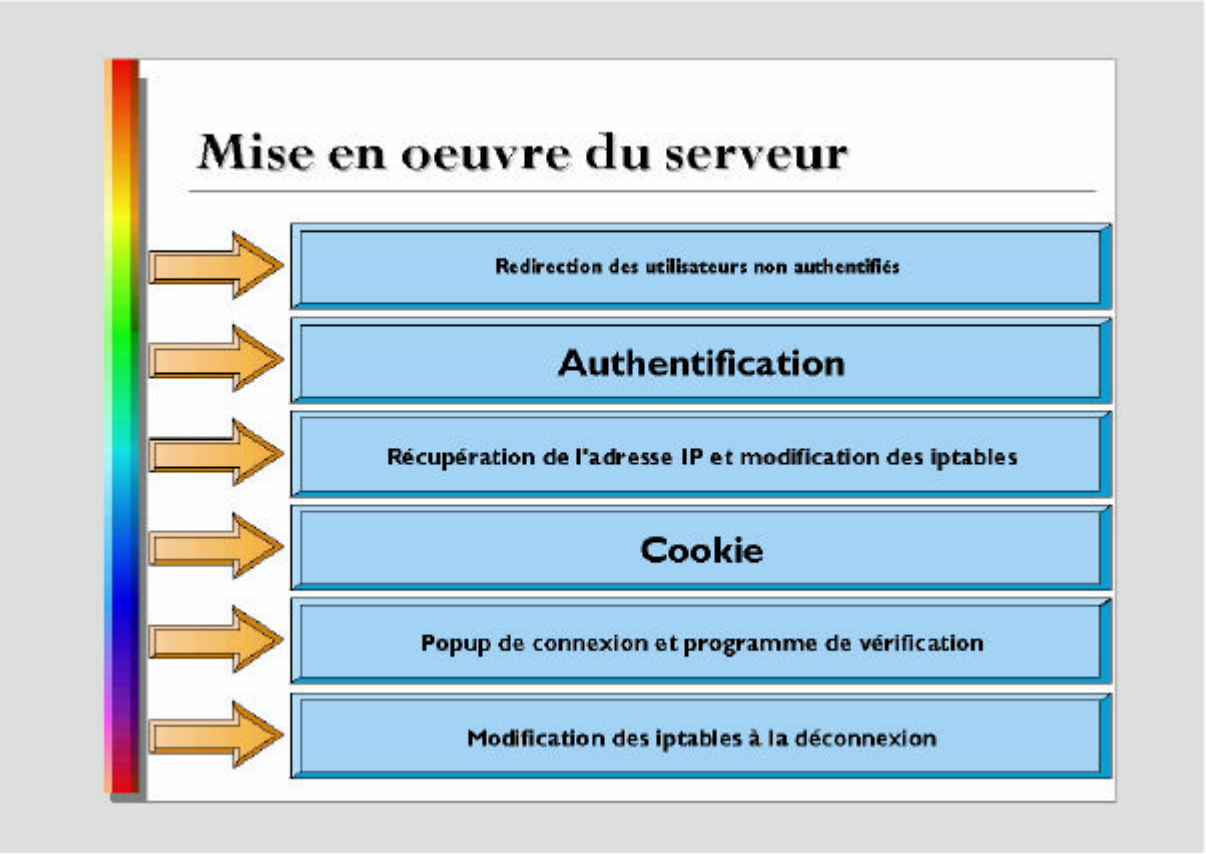
L'avantage d'un tel choix est que *iptables* est implanté directement dans le noyau Unix, il est donc plus rapide que n'importe quel programme de redirection "externe".

En plus des filtrages de sécurité décrits plus hauts, l'utilisation d'*iptables* permet d'effectuer la redirection de la manière suivante :

- toute requête HTTP venant de l'extérieur et à destination du réseau interne est interceptée par le serveur et redirigée vers la page d'authentification ;
- si l'authentification est acceptée, le serveur modifie ses *iptables* (par un script ou une commande shell par exemple) pour que les paquets provenant de cette machine soient acceptés et envoyés vers leur destination ;
- à la déconnexion de l'utilisateur ou si l'adresse IP n'est plus valide, les *iptables* sont à nouveau modifiées pour ne plus laisser passer les requêtes provenant de cette adresse IP, et une nouvelle authentification est alors demandée.

Dans ce cas, il ne doit pas y avoir de routeur entre le WiFi et le serveur (s'il y en avait un, il masquerait les adresses IP des machines par sa propre adresse IP). Le serveur peut jouer lui-même le rôle de la borne WiFi, il lui suffit de posséder une carte WiFi sur laquelle on aura branché une antenne disposant d'une bonne réception.

Le serveur doit avoir DHCP installé pour donner des adresses IP dynamiques aux machines des utilisateurs qui tentent de se connecter par le WiFi. L'inconvénient des adresses IP dynamiques est qu'un client peut s'insérer à la place d'un autre alors que l'adresse IP est toujours reconnue valide par le serveur. Pour éviter cela, il faut vérifier régulièrement (grâce à un timeout) les adresses IP et la présence d'un cookie à ces adresses, avec une période égale à la moitié de la période de renouvellement de l'adresse IP.



Remarques :

- L'avantage d'utiliser PHP/MySQL pour l'authentification est qu'on peut dans le même script utiliser PHP pour appeler des scripts ou des commandes shell, notamment pour modifier les *iptables* une fois l'authentification réussie. La fonction `shell_exec(string $cmd)` permet d'exécuter directement des commandes shell dans du PHP.
- On peut utiliser la même méthode pour modifier les *iptables* à la déconnexion.
- Le popup de connexion vérifie à chaque timeout l'adresse IP et la présence du cookie, et renouvelle ce dernier si besoin est.
- Le bon fonctionnement du système nécessite que le navigateur de l'utilisateur WiFi accepte les cookies et les popups.
- Ce principe permet d'étendre le serveur à la gestion d'autres protocoles réseaux, si la modification des *iptables* amène à "ouvrir" les ports dédiés à ces protocoles. On peut alors imaginer un système où chaque utilisateur est enregistré dans la base de données avec la liste des ports, donc des protocoles, qu'il a le droit d'utiliser sur le réseau. ("80, 110" ou "80, 110, 6667", etc ...)

Modification des *iptables* et récupération de l'adresse IP :

Les *iptables* sont organisées en "chaînes" de règles qui sont prises dans l'ordre où elles sont indiquées : si un paquet correspond à une règle, on l'applique, et si plus loin une seconde règle peut s'appliquer à ce paquet, elle n'est pas utilisée.

Il existe deux *iptables* principales :

- la table FILTER (filtrage par défaut) qui ne connaît que 2 commandes essentiellement : accepter le passage des paquets (ACCEPT) et rejeter les paquets (DROP).
- la table NAT dédiée à la modification des adresses. Elle agit en PREROUTING (paquets entrant dans le serveur) et en POSTROUTING (paquets sortant du serveur).

Pour répondre à la règle de sécurité du *"tout ce qui n'est pas explicitement autorisé est interdit"*, on définit des règles par défaut qui bloquent le trafic entrant et passant par le serveur (celui qui ne correspond à aucune règle prévue, et qui a donc des chances d'être de la connexion indésirable) :

```
-P INPUT DROP     //rien n'entre
-P FORWARD DROP   //rien ne passe
```

Par défaut, toute requête HTTP demandant une nouvelle connexion (state NEW), issue du WiFi et destinée au réseau interne est redirigée vers le serveur (du moins jusqu'à ce qu'il y ait une authentification). La règle qui s'applique est donc la suivante (règle indiquée dans la table NAT, chaîne PREROUTING) :

```
-t nat PREROUTING -i interface_WiFi -o interface_reseau_interne --dport 80\
-m state --state NEW -j DNAT --to-destination ip_serveur:80
```

Pour cela, il faut qu'il y ait préalablement dans la table FILTER une règle disant qu'on laisse passer ce qui passe par le serveur pour le protocole HTTP (port 80) (sinon, les paquets seront rejetés dès qu'ils tenteront de passer par le serveur et la règle NAT ne pourra jamais s'appliquer). Cette règle est :

```
FORWARD -i interface_WiFi -o interface_reseau_interne --dport 80 -m state \
--state NEW,ESTABLISHED -j ACCEPT
```

On ne laisse passer que les paquets demandant une nouvelle connexion et correspondant à une connexion établie (state ESTABLISHED), ce qui permet de rejeter les paquets correspondant à une connexion inconnue (state INVALID) et qui risquent donc d'être des paquets indésirables.

Une fois l'authentification faite, la connexion est établie et n'est alors plus dans les termes de la règle de redirection : le paquet HTTP peut alors passer sans être redirigé (puisque le FORWARD est toujours valide et que la règle de redirection ne s'applique plus à ce paquet). On peut alors associer à cet utilisateur des autorisations de se connecter au réseau sur certains ports (dans les 2 sens car il faut que les réponses du réseau interne soient également transmises) :

```
FORWARD -s IP_utilisateur_authentifié -o interface_resau_interne -m \
multiport --dport 110,6667 -m state --state ESTABLISHED -j ACCEPT

FORWARD -i interface_resau_interne -d IP_utilisateur_authentifié -m \
multiport --dport 110,6667 -m state --state ESTABLISHED -j ACCEPT
```

Dans cet exemple, l'utilisateur authentifié a accès, en plus de l'HTTP, au POP (port 110) et à l'IRC (port 6667) sur le réseau interne.

Si l'authentification échoue, le script PHP doit inclure un *die()* (coupure de connexion) afin que la prochaine connexion au réseau soit un NEW (l'utilisateur ne doit pas être considéré comme authentifié alors qu'il ne l'est pas).

Si l'utilisateur se déconnecte ou si le timeout expire alors que l'utilisateur n'est plus là, il faut enlever toutes les règles qui donnent accès au réseau à cet utilisateur (il devra se réauthentifier pour que ces règles soient à nouveau ajoutées). Pour cela, il faut générer les règles du type de la précédente qu'on a ajoutées pour cet utilisateur (à partir de son adresse IP et de la liste des ports autorisés) et effectuer un DELETE de la règle en question ; par exemple :

```
<?php
```

```
$regle= "FORWARD -s $ip_user -o interface_reseau_interne -m multiport \
--dport $port1,$port2 -m state --state ESTABLISHED -j ACCEPT";
//génération de la règle pour l'utilisateur

shell_exec(iptables -D $regle);     //effacement de la règle en question
?>
```

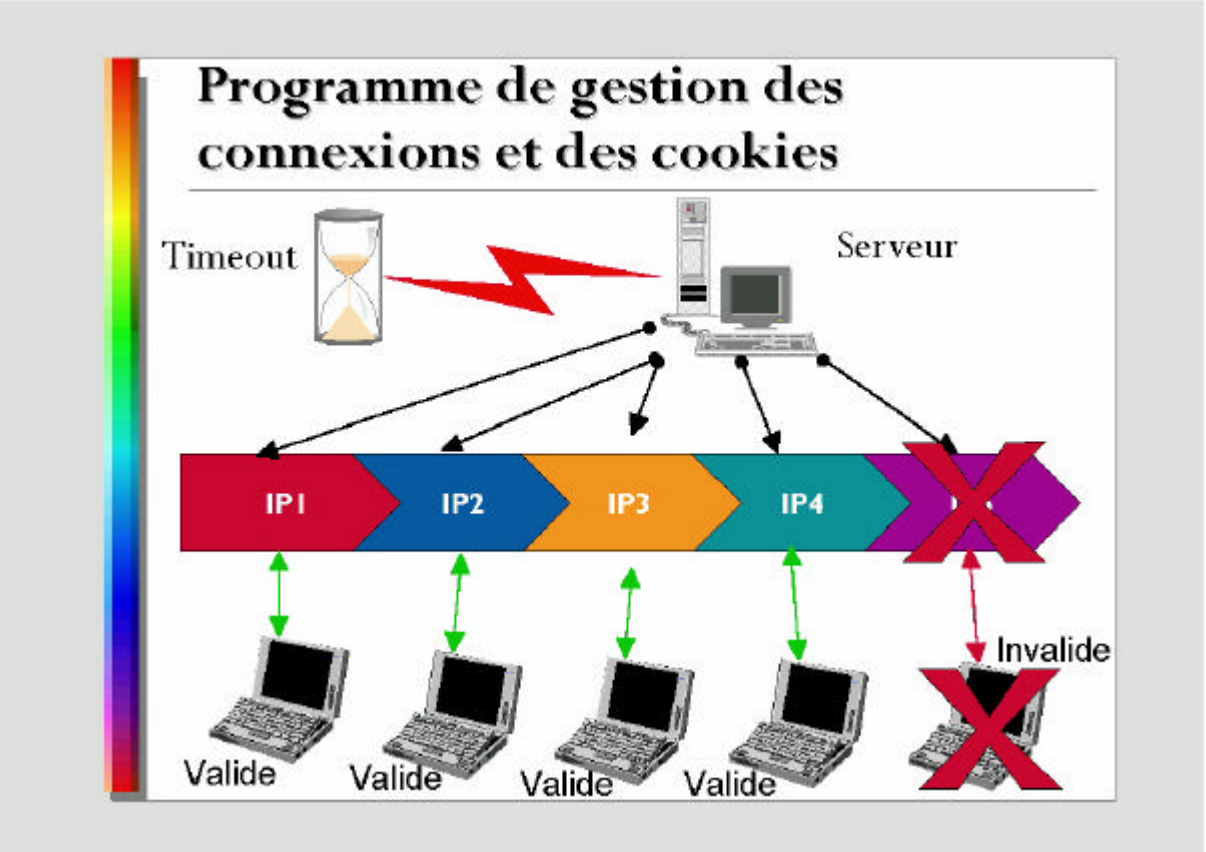
À noter qu'on peut utiliser le même type de script lors de l'ajout des règles (on génère la règle puis on l'ajoute).

Remarques :

- L'exécution des commandes de *iptables* nécessite que l'utilisateur soit *root*. Or, par mesure de sécurité, le serveur Apache ne possède généralement pas les droits du super-utilisateur. Il faut donner temporairement au serveur les droits nécessaires en utilisant la commande *sudo* (exécution en tant que super-utilisateur).
- Pour éviter les attaques de type "buffer overflow", on vérifie avant d'exécuter la commande la taille de l'adresse IP ainsi que son format. En temps "normal" (pas d'intrusion malveillante de la part d'un utilisateur), l'adresse IP vient d'une récupération en amont de la variable REMOTE_ADDR issue du serveur Apache, grâce à la fonction PHP `getenv("REMOTE_ADDR")`.

Vérification de l'adresse au timeout :

À chaque authentification, il faut enregistrer l'adresse IP de l'utilisateur authentifié afin de tenir à jour une liste (fichier ou base de données) des utilisateurs connectés par adresse IP.



La vérification de la validité de ces adresses se fait selon l'algorithme suivant :

Pour chaque adresse IP de la liste des utilisateurs connectés
{
    Vérifier la présence du cookie à cette adresse

Si le cookie est absent ou invalide (date limite expirée)
{
Couper l'accès au réseau (effacement de la règle d'iptables correspondant à cette adresse)
Enlever l'adresse IP de la liste des connectés
}
Si le cookie est bon
{
Renouveler le cookie pour une période
Renouveler le popup de connexion
}
}

Il faut également couper l'accès et enlever l'utilisateur de la liste des connectés en cas de déconnexion volontaire (clic sur un bouton du popup).

À chaque renouvellement du cookie, on peut enregistrer une trace de ce renouvellement, ce qui permet de savoir pour combien de périodes (et donc pour combien de temps) chaque utilisateur est resté connecté au réseau.

Les pages d'authentification

Page d'accueil (HTML simple)
Lors de la redirection vers le serveur, celui-ci affiche une page de ce type. Elle envoie les informations login/pass à une autre page nommée *auth.php* qui s'occupe de l'authentification.

Code de la page auth.php :
(seul le code PHP est indiqué ici)

```
//Données de connexion à la base de données

$host = nom_serveur_bdd; //l'hôte où se trouve la base de données n'est pas
obligatoirement le serveur lui-même

$user = nom_utilisateur_serveur_bdd;
$bdd = base;
$password = mot_de_passe_utilisateur_bdd;

//Connexion à la base de données

mysql_connect($host, $user,$password) or die("Erreur de connexion au
serveur");
mysql_select_db($bdd) or die("Erreur de connexion a la base de donnees");

//Vérification des données utilisateur

$query="SELECT login, pass, ports FROM base_utilisateurs WHERE
login='$login' AND pass='$pass'";
$result=mysql_query($query);
$numres=mysql_num_rows($result);

if($numres==0)
{
      echo "<H1>L'authentification a échoué !</H1>";
      echo "Votre login ou votre mot de passe est incorrect.";
      echo "<A href='accueil.htm'>Réessayer</A>";
      die();
}

Else
{
      //enregistrement de l'adresse IP pour les règles iptables et la
vérification périodique

      $ip_user=getenv("REMOTE_ADDR");

      // on peut insérer ici une vérification de la validité de l'adresse
IP (taille/format)

      $query="INSERT INTO base_ip (ip) VALUES ('ip_user')";
      Mysql_query($query);

      //récupération des ports autorisés (on suppose qu'ils sont dans la
base sous la forme "port1,port2, etc...")
```

```
      $row=mysql_fetch_row($result);
      $ports=$row[2];

      //génération de la règle pour l'utilisateur

      //ATTENTION : ceci ne marche pas pour le FTP, il faut utiliser des
règles spécifiques

      $regle= "FORWARD -s $ip_user -o interface_reseau_interne -m multiport
--dport $ports -m state --state ESTABLISHED -j ACCEPT";

// ici il faut ajouter quelque chose avec sudo pour donner au serveur
Apache les droits du super-utilisateur le temps du shell_exec

shell_exec(iptables -A $regle);     //ajout de la règle

      $regle= "FORWARD -i interface_reseau_interne -d $ip_user -m multiport
--dport $ports -m state --state ESTABLISHED -j ACCEPT";

// sudo de même

shell_exec(iptables -A $regle);     //ajout de la règle

Echo "<H1>Authentification acceptée !</H1>
Echo "Vous pouvez maintenant accéder au réseau.<BR><I>Le popup de connexion
garantit votre accès au réseau, ne le fermez pas !</I>"

}
```

La page *auth.php* contiendra également le lancement du popup de connexion (appel d'un script JavaScript par exemple) et l'enregistrement du cookie.

On peut apporter des améliorations à cette page. Par exemple, on peut inclure une RAZ des *iptables* si on tape comme login "reset" et comme mot de passe "reset" : *auth.php* réagirait alors en effaçant toutes les règles de redirection.