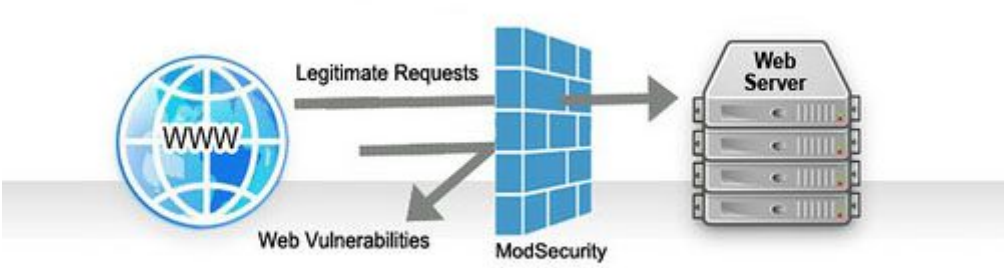
Recherches : Installation et configuration d'un firewall applicatif



Documentation : Protection d'une application web à l'aide d'un WAF (Web Application Firewall).

Auteur : Guillaume Orlando
Date : 26/03/2017

I. Introduction.

Un WAF, acronyme de “Web Application Firewall”, est un firewall applicatif ayant pour but de protéger des applications web potentiellements sujettes à des attaques de niveau haut de la couche OSI. Typiquement, ces attaques peuvent êtres des injections SQL, du cross-scripting (XSS), des buffers overflow, etc …

L’idée derrière un WAF est d’analyser l’ensemble des requêtes à destination d’une application web. A la manière d’un IDS, ces requêtes peuvent êtres comparées à une base de données de signatures d’attaques biens connues. Cette méthode nécessite cependant de mettre à jour régulièrement la base de signature utilisée, tout en comptant sur le fait qu’un attaquant ingénieux ne parvienne pas à exploiter l’application en utilisant des méthodes encore non répertoriées.

Le second mode de fonctionnement d’un firewall applicatif se compare à une whitelist : le fonctionnement normal de l’application est documenté, puis détaillé au WAF, de façon à signaler tout comportements sortants de la routine de l’application. Le niveau de sécurité est ici plus haut, mais la configuration plus fastidieuse.

Dans cette documentation, les deux méthodes de fonctionnements seront testés, sur un simple formulaire d’authentification écrit en PHP. Ce même formulaire est vulnérable à une injection SQL basique.

Nous allons donc commencer par déployer notre application web, qui sera volontairement vulnérable.

II. Installation des paquets nécessaires.

Avant toutes choses, installons les paquets nécessaire au bon fonctionnement de notre architecture. Un serveur web, un interpréteur PHP, et un serveur de base de données MySQL seront nécessaires :

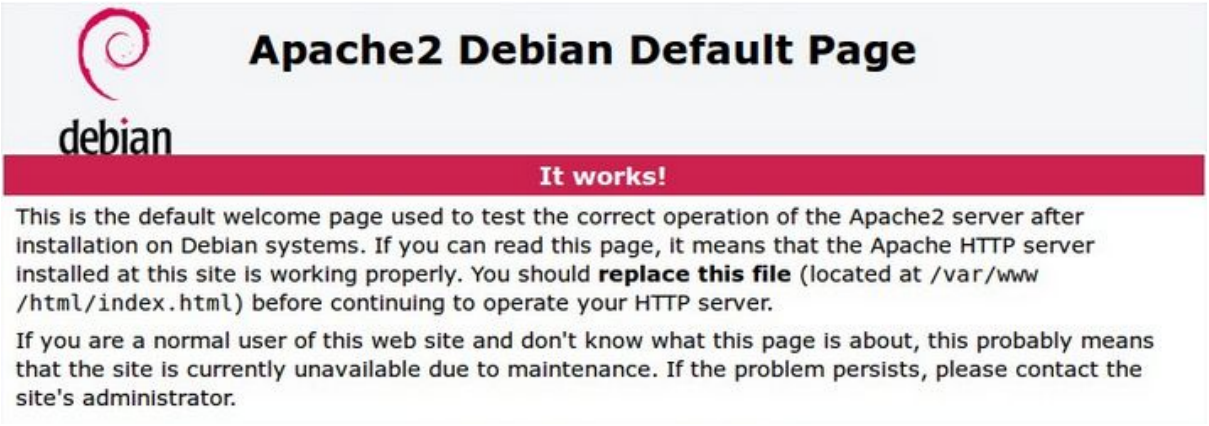
```
apt-get install apache2 libapache2-mod-php php php-mysql mysql-server
```

Il sera également nécessaire d’installer le paquet modsecurity pour apache, dont nous verrons l’utilitée plus tard :

```
apt-get install libapache2-modsecurity
```

II. Déploiement d'une application volontairement vulnérable.

Pour effectuer correctement notre test, nous allons avoir besoin d'une application délibérément vulnérable. Commençons par vérifier notre installation d'apache, en ouvrant un navigateur pointant sur l'IP de notre machine virtuelle :



Si la page par défaut d'apache s'affiche, nous pouvons continuer, et remplace l'index d'apache par notre formulaire PHP :

```
mv /var/www/html/index.html /var/www/html/index.php
echo “” > /var/www/html/index.php
```

Le formulaire vulnérable sera le suivant :

```
<html>
<body>
<?php
    if(isset($_POST['login']))
    {
        $username = $_POST['username'];
        $password = $_POST['password'];
        $con = mysqli_connect('localhost','root','password','sample');
        $result = mysqli_query($con, "SELECT * FROM `users` WHERE username='$username' AND password='$password'");
        if(mysqli_num_rows($result) == 0)
            echo 'Invalid username or password';
        else
            echo '<h1>Welcome !</h1>';
    }
    else
    {
?>
        <form action="" method="post">
            Username: <input type="text" name="username"/><br />
            Password: <input type="password" name="password"/><br />
            <input type="submit" name="login" value="Login"/>
        </form>
<?php
    }
?>
</body>
</html>
```

Prenons du temps pour détailler le fonctionnement de l'injection SQL à laquelle ce code est vulnérable.

III. Détail de la faille web exploitée.

En analysant plus précisément ce code, nous observons que celui-ci est trivial à exploiter dans le cadre d'une injection SQL. Le mécanisme d'authentification repose uniquement sur une comparaison des entrées saisies par l'utilisateur avec le contenu des identifiants préalablements entrées en base de données :

Le contenu de la table "users" correspondants aux champs utilisateurs est récupéré :

```
$result = mysqli_query($con, "SELECT * FROM `users` WHERE username='$username' AND password='$password'");
```

Ensuite, si la requête est valide (donc que l'utilisateur et le mot de passe existe), l'accès est autorisé :

```
if(mysqli_num_rows($result) == 0)
    echo 'Invalid username or password';
else
    echo '<h1>Welcome !</h1>';
```

Pour obtenir un accès, il faut donc que la requête soit valide. Ayant cette donnée en tête, observons plus précisément la requête SQL effectuée lors de notre authentification :

```
"SELECT * FROM `users` WHERE username='$username' AND password='$password'"
```

Essayons donc de la modifier de façon à créer une requête retournant toujours un résultat positif. Notre champ d'action se résume aux deux variables *$username* et *$password*, qui sont directements passé dans la requête de manière brute.

Nous allons donc être capable d'utiliser ce fait à notre avantage, et remplaçant le contenu de la variable *$username* par une instruction toujours valide et en terminant prématurément la requête.

Soit la requête craftée suivante :

‘ or 1=1 --

Nous allons tirer avantage du formulaire pour nous octroyer un accès illégitime en refermant le champ *$username* et en ajoutant une expression toujours vraie, à savoir *1=1*, puis en commentant la fin de la requête pour ne plus en tenir compte. Celle-ci va donc avoir la syntaxe suivante :

```
"SELECT * FROM `users` WHERE username='' or 1=1 --$username' AND password='$password'"
```

L'accès sera donc garanti, sans avoir eus recours à la méthode d'authentification initialement proposée.

IV. Test de l'application web et injection SQL.

Commençons par créer la base de donnée nécessaire au fonctionnement de l'application.

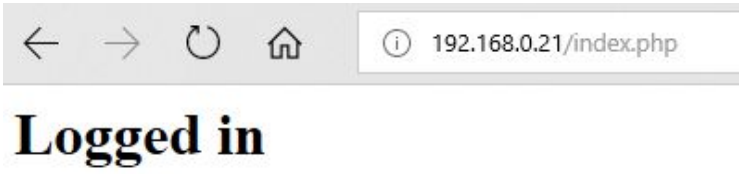
Nous allons nous authentifier en ligne de commande avec le super-utilisateur de l'instance MySQL défini lors de l'installation du paquet *mysql-server* :

```
mysql -u root -p
```

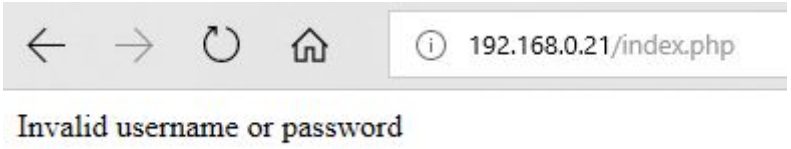
Nous allons ensuite créer la table requise à l'application, et créer un utilisateur légitime pour s'y connecter :

```
create database sample;
connect sample;
create table users(username VARCHAR(100),password VARCHAR(100));
insert into users values('user','pass');
quit;
```

En utilisant maintenant l'identifiant *"test"* et le mot de passe "*pass"* dans le formulaire, nous validons l'authentification légitimement :



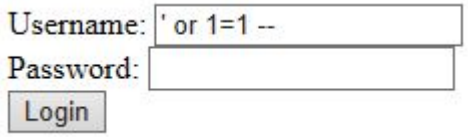
De même que des informations d'identifications erronées ne fonctionnent pas :



L'application dispose désormais d'une méthode d'authentification fonctionnelle.

Passons à son exploitation, en entrant la chaîne suivante dans le champ *username* :

```
‘ or 1=1 --
```



L'injection SQL est elle aussi fonctionnelle, puisque l'accès nous est désormais octroyé de façon détournée.