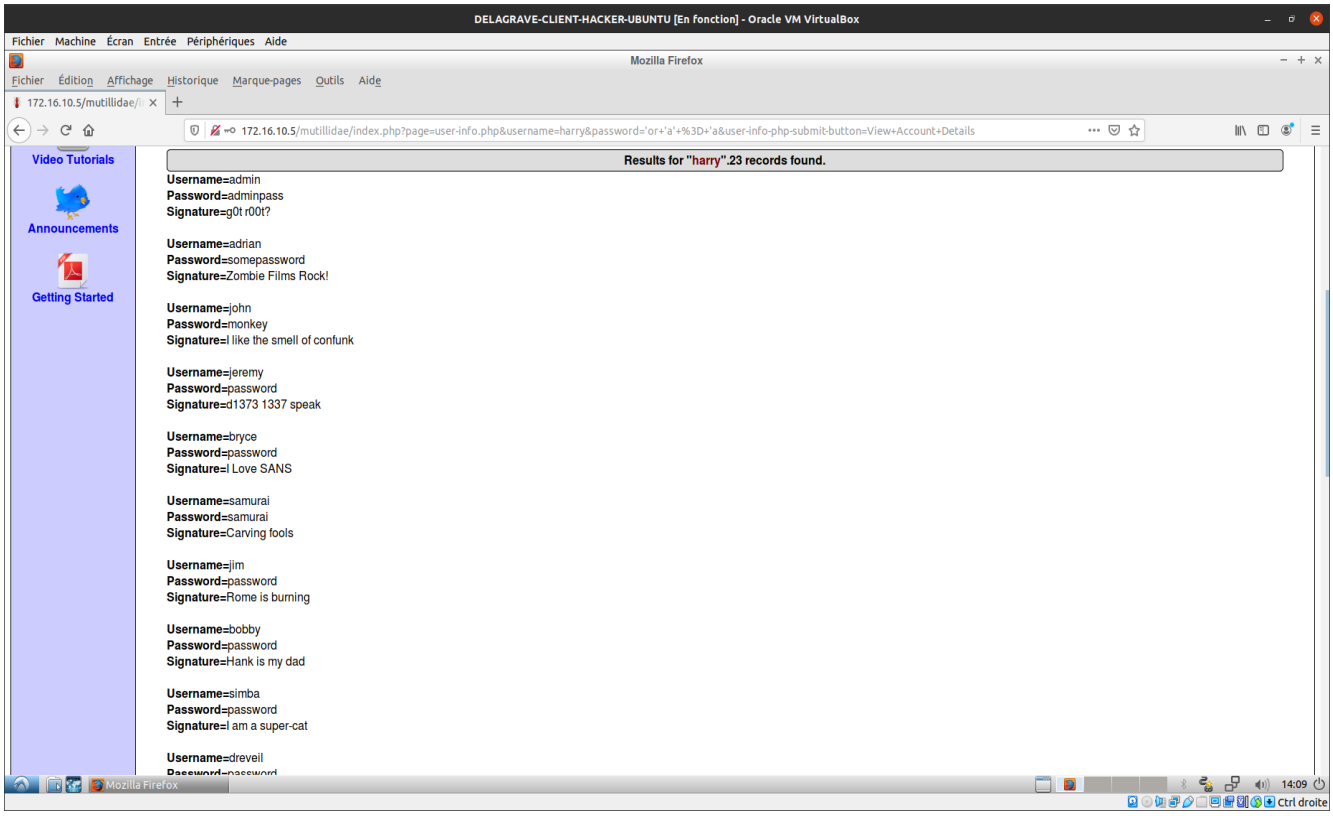
TAN
Kenthyvuth
1BTSB

TP04 - Protéger une application Web

ÉTAPE 2 : attaque par injection SQL

1) **Que constatez-vous ? Montrer le résultat avec une copie d'écran et commentez le.**

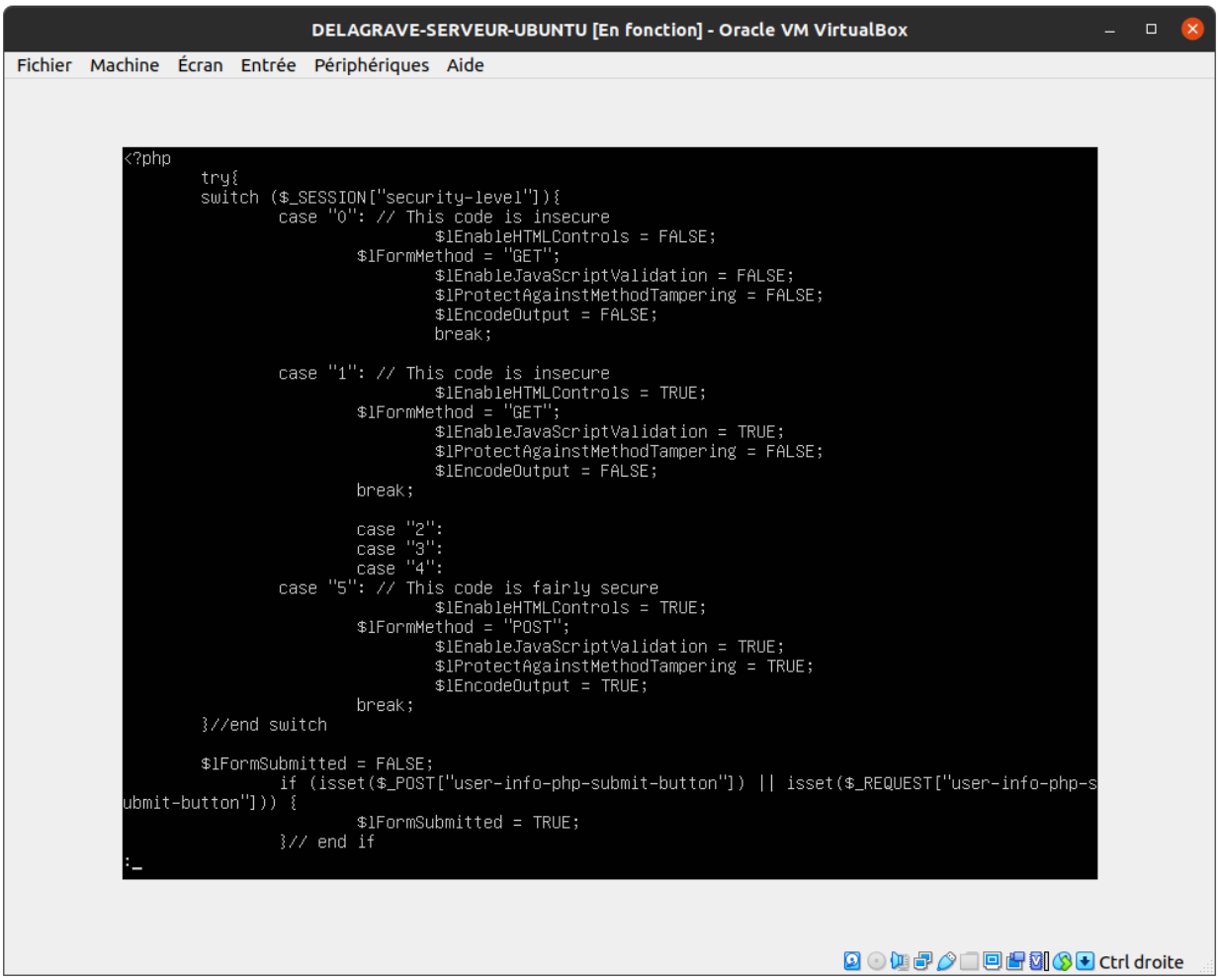


(Capture d'écran des résultats après authentification)

Après avoir saisi le nom d'utilisateur et le mot de passe, la page nous affiche la liste des utilisateurs inscrits sur le site suivis de leur mot de passe et de leur signature. Ceci s'explique par le le bout de code que l'on a inséré dans le champ "password", qui est un code SQL qui nous retourne alors des informations à caractère personnel.

ÉTAPE 3 : mise en place d'une contre-mesure

2) **Que constatez-vous dès le début du code ? Montrer le résultat avec une copie d'écran et commentez le.**

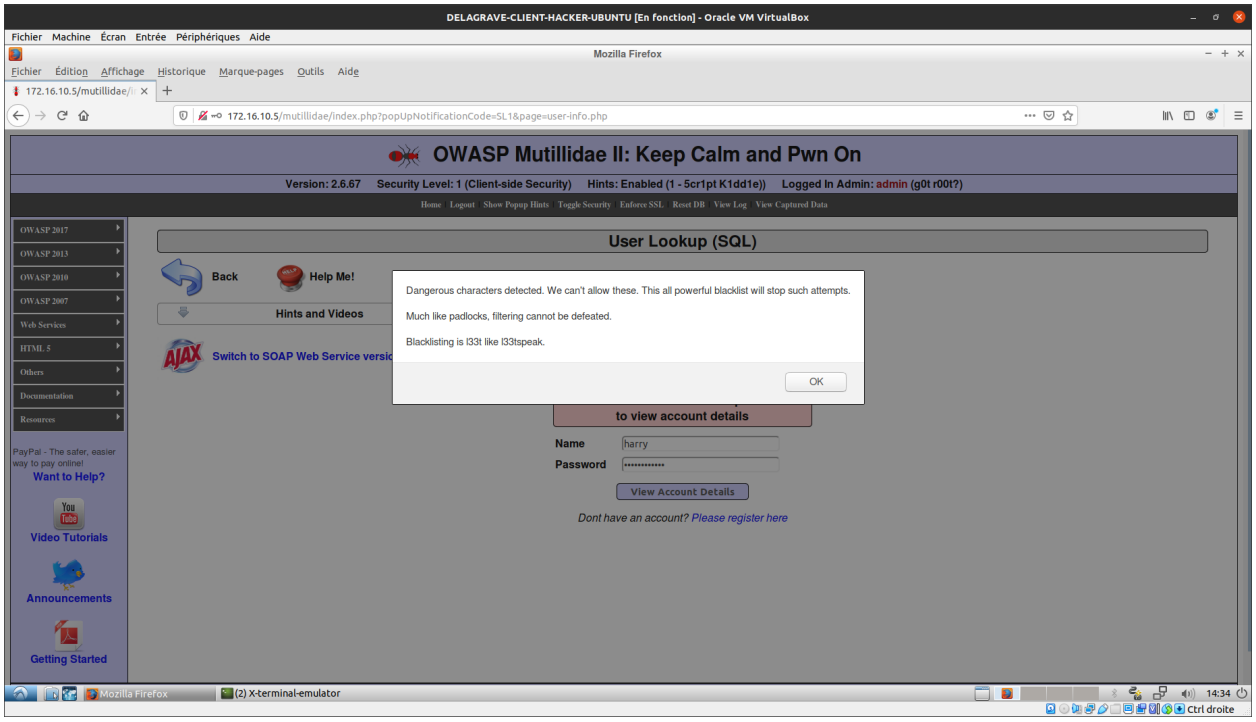



```
<?php
        try{
        switch ($_SESSION["security-level"]){
                case "0": // This code is insecure
                                $lEnableHTMLControls = FALSE;
                        $lFormMethod = "GET";
                                $lEnableJavaScriptValidation = FALSE;
                                $lProtectAgainstMethodTampering = FALSE;
                                $lEncodeOutput = FALSE;
                                break;

                case "1": // This code is insecure
                                $lEnableHTMLControls = TRUE;
                        $lFormMethod = "GET";
                                $lEnableJavaScriptValidation = TRUE;
                                $lProtectAgainstMethodTampering = FALSE;
                                $lEncodeOutput = FALSE;
                        break;

                        case "2":
                        case "3":
                        case "4":
                case "5": // This code is fairly secure
                                $lEnableHTMLControls = TRUE;
                        $lFormMethod = "POST";
                                $lEnableJavaScriptValidation = TRUE;
                                $lProtectAgainstMethodTampering = TRUE;
                                $lEncodeOutput = TRUE;
                        break;
        }//end switch

        $lFormSubmitted = FALSE;
                if (isset($_POST["user-info-php-submit-button"]) || isset($_REQUEST["user-info-php-s
ubmit-button"])) {
                        $lFormSubmitted = TRUE;
                }// end if
:_
```

(Extrait du code user-info.php)

On remarque en début de code, un switch qui détecte le niveau de sécurité présent sur le site Mutillidae. Ce niveau de sécurité est évalué par les variables “caseX” avec X variant entre 0, 1 et 5, soit du niveau le moins sécurisé au niveau le plus sécurisé. Selon le niveau détecté, le code exécutera différentes fonctions (évaluées par les valeurs TRUE et FALSE).

3) **Rejouer la même attaque par injection SQL que précédemment. Que constatez-vous ? Montrer le résultat avec une copie d'écran et commentez le.**



(Test de connexion avec une sécurité de niveau 1)

On a changé manuellement le niveau de sécurité en le passant à 1 (cf. voir ci-dessus). En rejouant la même attaque donc en retapant le nom et le mot de passe, une fenêtre s'affiche. Elle annonce que "des caractères dangereux ont été détectés". Il s'agit en effet des caractères spéciaux " ' " et " = " qui ont été renseignés dans une liste noire appelée "**lUnsafeCharacters**". Le site nous refuse alors la connexion avec ces identifiants.

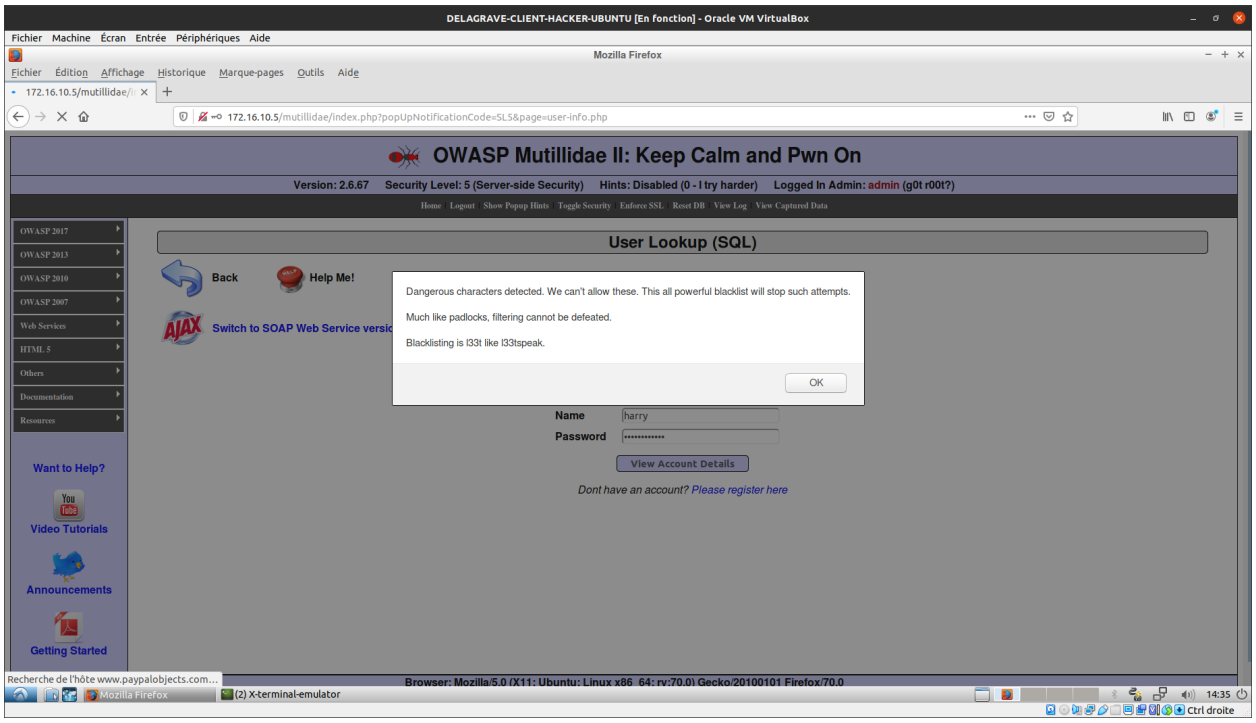
```
var lUnsafeCharacters = /[`~!@#$%^&*()-_=+\[\]{}\\|;':",./<>?]/;
```

(La liste noire en question)

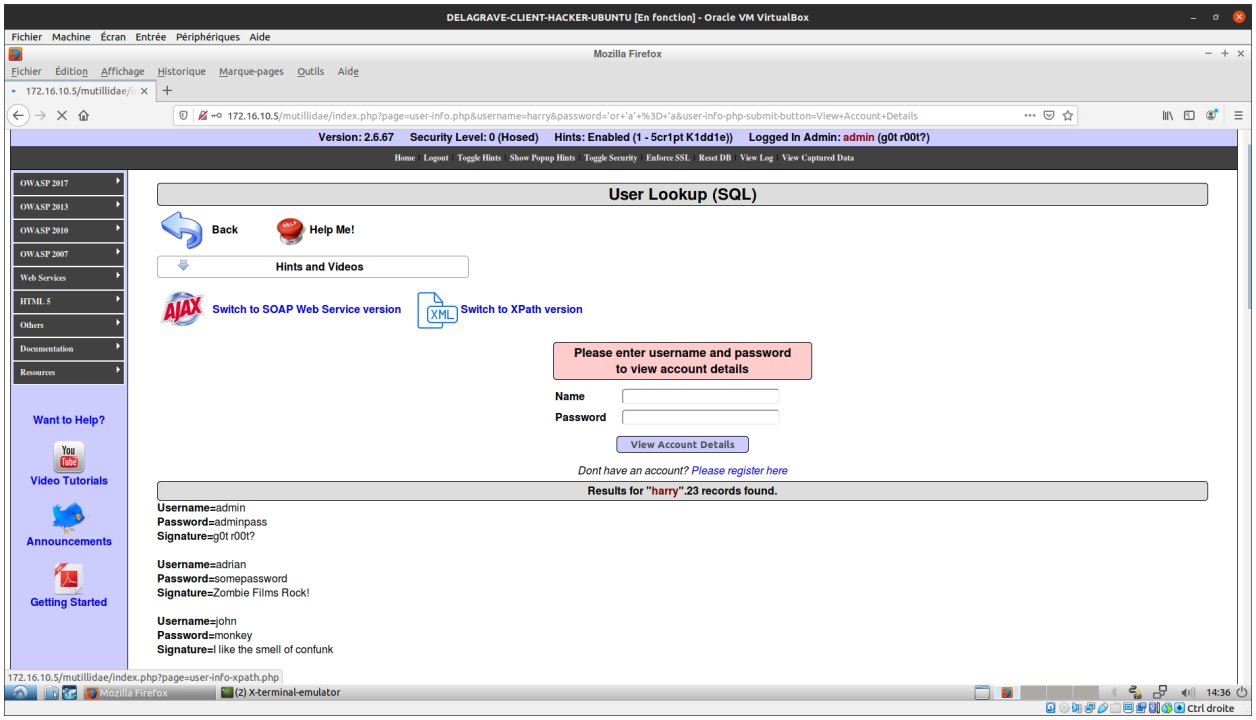
- **Rejouez simultanément les deux attaques, niveau 0 (non sécurisé) et niveau 5 (sécurisé).**
- **Dans chaque cas, conservez le code de la page web reçue.**

- **Comparez les deux pages web reçues, montrez les différences avec une copie d'écran et commentez le en répondant aux questions suivantes :**

4) **Quelle partie du code permet d'éviter l'injection SQL ? Que pouvez-vous en déduire en matière de bonnes pratiques de codage sécurisé ?**



(Test de connexion avec une sécurité de niveau 5)



(Test de connexion avec une sécurité de niveau 0)

D'après les deux captures d'écran (cf. voir au-dessus), on remarque bien que selon le niveau de sécurité, le code PHP nous renvoie une réponse différente. Lorsque le niveau de sécurité est à 5, tout comme pour le niveau 1, un message nous avertit que des caractères spéciaux ont été utilisés pour l'authentification.

La partie du code qui permet ainsi d'éviter l'injection SQL est représentée par le switch et l'ensemble des "caseX" dont chaque "case" comprend des variables qui renvoient à des codes de sécurité. On voit qu'il y a des variables ayant pour valeur "FALSE", ce qui signifie qu'une sécurité est désactivée.

En termes de bonne pratique dans le codage sécurisé, concernant la faille SQL, il faut établir une liste noire avec les caractères appropriés pour les pirates. Les valeurs booléennes qui sont attribuées aux différents codes de sécurité doivent avoir une attention particulière lors de l'écriture du programme.
De façon générale, il faut également mettre des commentaires afin d'expliquer le fonctionnement du code. En l'occurrence ici, expliquer les niveaux de sécurité et le code exécuté lorsque le programme entre dans tel niveau.