Práctica 2. Autenticación

Taller de desarrollo seguro con PHP

Mayo 2019

1. Objetivo de la práctica

El alumno deberá asegurar el sitio que permite la autenticación de un usuario a través de programación segura en PHP. Se deberá proteger el componente mediante las siguientes técnicas:

- Usar password verify para la autenticación
- Validación de datos.
- Sentencias preparadas.
- Sanitizacion de datos.
- Proteger contra fuerza bruta, limitando la cantidad de acceso y bloqueando cuentas.

2. Uso de password verify

Para comprobar que una contraseña ingresada coincide con el Hash almacenado en el registro de usuario se puede hacer uso de la función *password_verify()*. Sólo se pasa como primer parámetro la contraseña recibida y como segundo el Hash almacenado.

Listing 1: Comprobación de Hash

```
// Sin password_verify
$sql = "SELECT␣usuario_id,␣nombre,␣apellido
FROM␣usuarios␣WHERE␣email␣=␣'$email'␣and␣password='$password'";
$resultado =  $mysqli->query($sql)
//Con password_verify
$sql = "SELECT␣usuario_id,␣nombre,␣apellido,␣password
FROM␣usuarios␣WHERE␣email␣=␣'$email'";
$resultado =  $mysqli->query($sql)
if(!password_verify($password, $usuario['password'])){
        $mensaje = "Usuario␣o␣contrasena␣equivocados";
}
```

3. Validación de datos

Se debe validar que los campos no estén vacíos y que el correo electrónico sea un correo válido. Para eso, se puede implementar lo siguiente:

Listing 2: Validación de datos

```
//Funcion para validar que no haya datos vacios
function requerido($valor)
{
    $trimmedValue = trim($valor);
    return isset($trimmedValue) && $trimmedValue !== "";
}
//Funcion que valida el formulario
function formValidation($email,$password){
    //Se valida que los datos no esten vacios
    if (!requerido($email) || !requerido($email)) {
        return false;
    }

    //Se revisa que sea un email valido
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        return false;
    }

    return true;
}

//Antes de realizar cualquier accion se validan datos
    if (formValidation($email,$password)) {
        //Resto del codigo
    }else{
        $errorMessage = "Error en la validacion de los datos";
    }
```

4. Sentencias preparada.

Para evitar cualquier inyección de código SQL se deben preparar las sentencias como se ilustra a continuación:

Listing 3: Preparación de sentencias

```
//No preparadas
$sql = "SELECT usuario_id, nombre, apellido, password
FROM usuarios WHERE email = '$email'";
$resultado =  $mysqli->query($sql);
$usuario = $resultado->fetch_assoc();
```

```php
//Preparadas
$sql = 'SELECT usuario_id, nombre, apellido, password FROM usuarios WHERE email = ?';
$data = $mysqli->prepare( $sql );
$data->bind_param( 's', $email);
$resultado = $data->execute();
$data->bind_result($usuario['id'],$usuario['nombre'],$usuario['apellido'],$usuario['password']);
$data->fetch();
```

5. Sanitizacion de datos

Otra forma de proteger una aplicación web es através de la *sanitización* de datos. Este proceso consiste en la eliminación o escapado de caracteres no deseados en la información que se recibe. Se utiliza principalmente para evitar ataques XSS y de inyección de código SQL.

PHP nos provee de funciones que nos permiten sanear diferentes tipos de cadenas de información. A continuación se muestran las funciones que se utilizarán:

- *htmlentities()*. Convierte todos los caracteres aplicables a entidades HTML
- *mysqli_real_escape_string()*. Escapa los caracteres especiales de una cadena para usarla en una sentencia SQL. Evita errores en caso de estar presentes estos caracteres.

Para ver el catálogo completo de funciones que proporciona PHP, se puede aceder a su documentación a través de la siguiente liga

Listing 4: Saneado de datos

```php
//Codificacion de caracteres html
$email = htmlentities($email);
//Se escapan los caracteres especiales de una cadena para usarla en una sentencia SQL
$email = $mysqli->real_escape_string($email);
```

6. Protección contra fuerza bruta

Para proteger una aplicación web ante un ataque de fuerza bruta se deben limitar el número de intentos para el inicio de sesión. Esto frustrará el ataque, pues no permitirá probar todas las combinaciones posibles para hallar las credenciales adecuadas. En el ejemplo que se mostrará a continuación se utilizaron dos columnas en la base de datos para almacenar el último intento de inicio de sesión y el número de intentos fallidos.

Lo primero que se debe verificar es el número de intentos fallidos que el usuario ha acumulado, el tiempo transcurrido en caso de ser más del número máximo permitido para conforme a eso bloquear el inicio de sesión o no.

```
failedLogin($mysqli, $total_failed_login, $lockout_time, $email){
    //Se revisan sesiones fallidas y ultimo acceso
    $sql='SELECT intentos_fallidos, ultimo_acceso FROM usuarios WHERE email = ?';
    $data = $mysqli->prepare( $sql );
    $data->bind_param('s', $email);
    $data->execute();
    $data->bind_result($usuario['intentos_fallidos'], $usuario['ultimo_acceso']);
    $data->fetch();
    $data->close();

    //Si el numero de intentos fallidos es mayor al permitido...
    if ($usuario['intentos_fallidos'] > $total_failed_login) {
        // Se calcula si el usuario puede volver a conectarse
        $last_login = strtotime($usuario['ultimo_acceso']);
        $timeout    = $last_login + ($lockout_time * 60 * 60);
        $timenow    = time();
        // Si el tiempo suficiente no ha pasado, se bloquea
        if( $timenow < $timeout ) {
            return false;
        }else{
            return true;
        }
        return false;
    }

    return true;
}
```

Posteriormente, se debe validar que la cuenta no este bloqueada.

```
if (failedLogin($mysqli, $total_failed_login, $lockout_time, $email)) {
    //Resto del codigo
}else{
    $errorMessage = "Cuenta bloqueada por exceso de intentos fallidos";
}
```

Sin embargo, en caso de que el usuario se autentique con éxito en alguno de los tres intentos o después de ser bloqueado, se debe limpiar su cuenta de intentos de inicio de sesión fallidos.

```
//Se limpia registro de intentos fallidos
$sql='UPDATE usuarios SET intentos_fallidos = "0" WHERE email = ?';
$data = $mysqli->prepare( $sql );
$data->bind_param('s', $email);
$data->execute();
```

Por otra parte, si continua haciendo intentos fallidos, se deben contabilizar.

```
//Se suma un intento fallido
$sql= 'UPDATE usuarios SET intentos_fallidos = (intentos_fallidos + 1) WHERE email = ?';
$data = $mysqli->prepare($sql);
$data->bind_param('s', $email);
$data->execute();
```

Referencias

[1] https://www.php.net/manual/es/