Security and access control

DWES UD5.1

1.- User authentication and access control

For an access system to be effective, it is necessary to make http connections using a secure protocol: **https**.

To use https you need a valid certificate signed by a **trusted entity**.

In this way, the communication between the client and the server will be fully encrypted.

1.- User authentication and access control

Techniques such as electronic user certificates can be used for user authentication. An example is the electronic DNI. This authentication method is known as something that **the user has**.

To implement these systems, you must have a high level of knowledge of how they work and users must also have the appropriate hardware (ID card reader).

This is why it is more common to use a method known as something that the **user knows**, such as the **username and password set**.

2.- HTTP authentication

The browser should display a window to enter credentials.

An access control list is used to define users and their passwords.

If the user is not authenticated, the server responds with the code 401: "Unauthorized access."

In Apache, htpasswd is used, a file is created where the users and their (encrypted) passwords are stored. We will see it in DAW.

2.- HTTP authentication

With PHP you can access the credentials entered in said file using the global **$_SERVER array**:

- $_SERVER['PHP_AUTH_USER']
- $_SERVER['PHP_AUTH_PW']
- $_SERVER['AUTH_TYPE']

This method requires the introduction of users one by one in the file, which for large applications is not the best option, and it also makes it difficult for users to self-register in web applications.

<u>This method is not recommended and we are not going to see it</u>.

3.- Authentication by PHP

A better solution is to store the credentials in external storage, such as a database or a LDAP.

The credentials may be isolated in their own database or within a table to which only a specific user has permissions. They must be stored encrypted.

During the login process, the password entered will be encrypted and compared with the password stored in the database. If they match, the login will be valid; if they do not match, you will be informed.

4.- Password encryption

Hash

A **hash function** converts an input into a string of finite length.

The use of hashes in passwords allows the stored password to not be known in the event of data theft.

Hashing only protects stored passwords. It does not protect the process during registration or identification, for that you have to use https and avoid code injection.

4.- Password encryption

Hash

Historically the most used hashing algorithms have been **md5** y **sha1**. Nowadays they can be broken by brute force in a relatively short time.

In PHP 4 the **crypt** function is introduced to create password hashes.

Starting with PHP 5.5, password_hash is included. This API supports hashes created with the **crypt function**.

The use of password_hash is recommended.

5.- Function: crypt

To create a hash with crypt it is done as follows:

```
$hash = crypt($password, $salt);
```

A **salt** is a piece of data that is randomly calculated and used to generate the hash. Makes hashes harder to crack.

When calculating the hash, the cost is taken into account, this is the degree of complexity when applying the encryption algorithm. By default the cost is 10, but it should be decreased or increased depending on the hardware.

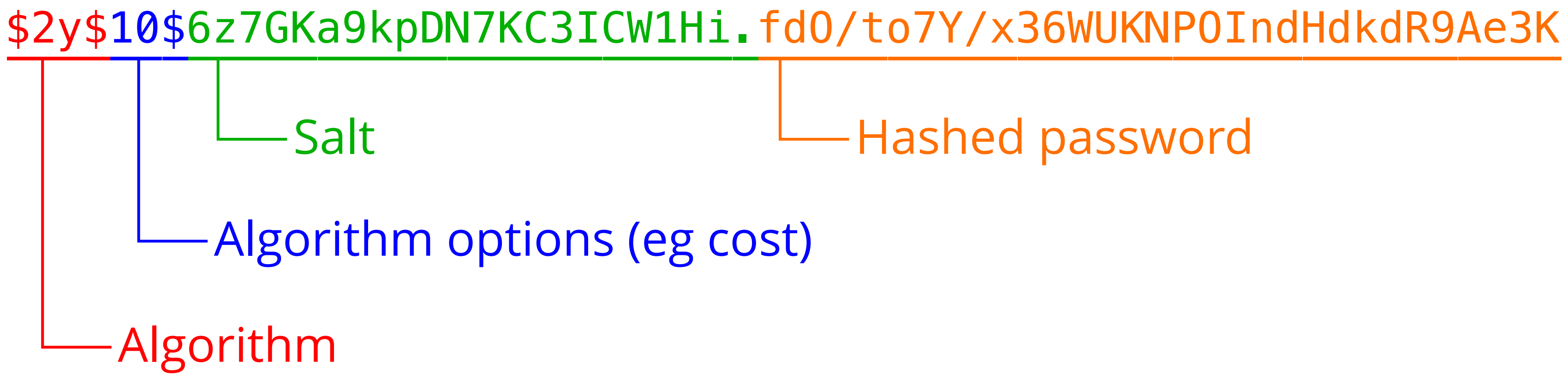
A very low cost creates less secure hashes.

Too high a cost will slow down the server.

5.- Function: crypt

Stored hash

Hashes store the following data:



```
$2y$10$6z7GKa9kpDN7KC3ICW1Hi.fdO/to7Y/x36WUKNPOIndHdkdR9Ae3K
```

5.- Function: crypt

Creating a password hash

The use of the **blowfish** algorithm is recommended.

The salt for blowfish should start with one of the following options:

- $2a$
- $2x$
- $2y$ → recommended for safety reasons

Next, the two-digit cost followed by $

Finally the salt, 22 characters from the set: a-z A-Z 0-9 . /

5.- Function: crypt

Creating a random hash of a password

```
$pass = 'mi_Contraseña25';

$salt = '$2y$12$';  // blowfish with complexity 12
$salt_chars = array_merge(range('A','Z'), range('a','z'), range(0,9), array('/', '.'));
for($i=0; $i < 22; $i++) $salt .= $salt_chars[array_rand($salt_chars)];

// use of crypt function
$hash = crypt($pass, $salt);
echo $hash;

// $2y$12$dqkCw9qJGDECKaG9aWj.deYbMI59h9FQVvt.4EGCkUKaNN00yaL6W
//This would be what is saved in the database or what is used to compare
```

5.- Function: crypt

Calculation of the optimal cost for hardware:

```
$salt = '';
$salt_chars = array_merge(range('A','Z'), range('a','z'), range(0,9), array('/', '.'));
for($i=0; $i < 22; $i++)
        $salt .= $salt_chars[array_rand($salt_chars)];

$timeTarget = 0.05;             // 50 milliseconds - acceptable time
$coste = 8;
do {
        $coste++;
        $saltOK = '$2y$'. $coste .'$'. $salt;
        $inicio = microtime(true);
        crypt('test', $saltOK);
        $fin = microtime(true);
} while (($fin - $inicio) < $timeTarget);

echo 'Coste apropiado encontrado: ' . $coste;
```

5.- Function: hash_equals

Checking a password

The check will be performed using the hash_equals function, which will check the stored hash with the hash of the entered password.

For the hash to be the same, the same salt should be used, but the salt is random, is not generated in the application and is not stored as such in the database.

The crypt function allows you to use the same salt used previously if a hash is passed as the salt parameter.

```
hash_equals($hash_bbdd, crypt('mi_Contraseña81', $hash_bbdd));
```

5.- Function: hash_equals

Checking a password

```
if(hash_equals($hash, crypt('mi_Contraseña25', $hash)))
		echo 'contraseña correcta';
else
		echo 'contraseña incorrecta';
```

6.- Function: password_hash

The function password_hash creates a new password hash using a strong one-way hashing algorithm. There are different supported algorithms.

The parameters it receives are the password, the encryption algorithm and an array of algorithm options.

The function password_verify verifies that a password matches a hash. The parameters it receives are the password and a hash. It returns the algorithm, cost and salt as part of the returned hash. Therefore, all information that's needed to verify the hash is included in it.

Password_hash and password_verify are the recommended functions.

6.- Function: password_hash

```
$pass = 'mi_Contraseña81';
$hash = password_hash($pass, PASSWORD_DEFAULT);
echo $hash;

if (password_verify($pass, $hash) {
    echo 'La contraseña es válida';
} else {
    echo 'La contraseña no es válida';
};
```

6.- Función: password_hash

Calculation of the optimal cost for hardware

```
        $timeTarget = 0.05;          // 50 milisegundos - tiempo aceptable
        $coste = 8;
        do {
                $coste++;
                $inicio = microtime(true);
                password_hash('test', PASSWORD_BCRYPT, ['cost' => $coste]);
                $fin = microtime(true);
        } while (($fin - $inicio) < $timeTarget);

    echo 'Coste apropiado encontrado: ' . $coste;
```

Exercise

Create the 'user_table' table in the database:

```
CREATE TABLE `discografia`.`tabla_usuarios` (
`id` INT NOT NULL AUTO_INCREMENT,
`usuario` VARCHAR(50) NOT NULL ,
`password` VARCHAR(255) NOT NULL ,
PRIMARY KEY (`id`)) ENGINE = InnoDB;
```

Inserts several users into the table, by hand or by creating a php script to do it.

Create a php script that shows a **login.php** screen. When entering username and password it will be compared with the content of the previous table. If correct, it will show the message 'Login successful'. If not, it will show 'Login failed'