

**Institute of Technology Carlow Software Development**

**Secure App Development**

**PHP Login System**

PART II

|  |  |  |
| --- | --- | --- |
| **Date: 12/03/2018** |  |  |
| **Subject: Secure App Dev** |  |  |
| **Lecturer: Richard Butler** |  | Garry Byrne C00120055 |

## Behaviours Logged

function normalLog($label,$logUser,$logUserAgent,$timeStamp,$query1,$query2,$query3,$query4,$logIP)

{

$k = file\_get\_contents('k.txt');

$message = "Action: ".$label."?????".

"Username: ".$logUser."?????".

"UserAgent: ".$logUserAgent."?????".

"Timestamp: ".$timeStamp."?????".

"Query Used 1: ".$query1."?????".

"Query Used 2: ".$query2 ."?????".

"Query Used 3: ".$query3."\r\n"."?????".

"Query Used 4: ".$query4."\r\n"."?????".

"IP Address: ".$logIP."?????";

$sanitizedLogVariable = filter\_var($message, FILTER\_SANITIZE\_STRING);

$cipher = "aes-128-gcm";

$key = $k;

$ivlen = openssl\_cipher\_iv\_length($cipher);

$iv = openssl\_random\_pseudo\_bytes($ivlen);

$ciphertext = openssl\_encrypt($sanitizedLogVariable, $cipher, $key, $options=0, $iv, $tag);

//store $cipher, $iv, and $tag for decryption later

$encrypted = $ciphertext. "@@@@@" . $iv . "@@@@@" . $tag . "#####";

$myfile = fopen("errorLog.txt", "a") or die("Unable to open file!");

fwrite($myfile, $encrypted);

fclose($myfile);

return $randomString;

}

Every Action (Login, Logout, forgot password etc) are logged along with SQL queries and if the database cannot connect. The above function is used to insert a log entry into the log file.

The below function is used to insert a log entry into the log file for failed SQL querys..

function SQLLog($label,$location,$error,$timeStamp)

{

$k = file\_get\_contents('k.txt');

$message = "Action: ".$label."?????".

"Location: ".$location."?????".

"Error: ".$error."?????".

"Timestamp: ".$timeStamp."?????";

$sanitizedLogVariable = filter\_var($message, FILTER\_SANITIZE\_STRING);

$cipher = "aes-128-gcm";

$key = $k;

$ivlen = openssl\_cipher\_iv\_length($cipher);

$iv = openssl\_random\_pseudo\_bytes($ivlen);

$ciphertext = openssl\_encrypt($sanitizedLogVariable, $cipher, $key, $options=0, $iv, $tag);

//store $cipher, $iv, and $tag for decryption later

$encrypted = $ciphertext. "@@@@@" . $iv . "@@@@@" . $tag . "#####";

$myfile = fopen("errorLog.txt", "a");

fwrite($myfile, $encrypted);

fclose($myfile);

}

I have 2 different functions because I need to record different information(Actions and SQL errors)

## Encryption

All data in the database is hashed by using the function crypt () which returns a hashed string using the standard Unix DES-based algorithm or alternative algorithms that may be available on the system.

I use the following function to compare what the user enters, to what is returned from the database.hash\_equals($returnedPassword, crypt($mypassword , $returnedPassword)).

This takes the users input and uses the stored salt to hash it and if the values match TRUE is returned, otherwise FALSE will be returned.

I used openssl\_encrypt() for encrypting the error log entry’s and openssl\_decrypt() to decrypt the error logs entry’s. The key for decryption is saved in k.txt, this is a proof of concept, it would not be like this in real lfe.

$ciphertext = openssl\_encrypt($plaintext, $cipher, $key, $iv, $tag);

$ciphertext = openssl\_decrypt($plaintext, $cipher, $key, $iv, $tag);

**cipher**

The cipher method. For a list of available cipher methods, use openssl\_get\_cipher\_methods().

**key**

The key.

**iv**

A non-NULL Initialization Vector.

**tag**

The authentication tag passed by reference when using AEAD cipher mode (GCM or CCM).

In the log file entry’s are split by “#####”. $ciphertext, $iv and $tag are split by “@@@@@”. And the individual components of the message are split by “?????”.

## Token Generation

The way I generated the token was by using the function

string **random\_bytes** ( int $length )

random\_bytes() generates an arbitrary length string of cryptographic random bytes that are suitable for cryptographic use. I set the length to 50 as I thought this would be secure. random\_bytes() comes as standard with PHP7.

The function returns the token as a binary value so I use bin2hex() to turn this binary value into a hexadecimal value. The token is then sent to the database and store in a table under the column name resetToken. The token is only active for 5 minutes, after this time has elapsed without a successful password reset the token is then invalid. The user will then have to resubmit a password reset.

I pass the token to the user by using a session variable to pass the token to a new page, in a real life scenario I would not do it this way. Instead I would send the reset token by email but as I am using XAMPP this method is not achievable.