Mar 01, 2015
CheckoutCrypto Canada

Introduction to PHP Encryption

By: Grant Hutchinson

Mar 01, 2015
CheckoutCrypto Canada

Index

http://php.net/manual/en/function.crypt.php

Mar 01, 2015
CheckoutCrypto Canada

Overview

Prerequisites: PHP knowledge, apache server with php5 installed and configured.

Hashing

Converting a string into a hashed, random, string or integer, that cannot be reversed(nearly impossible).

Encrypt

The process by which a string is compiled using a standard leading algorithm, into a string or integer, that can only be reversed in the instance of validation with the right: algorithm, salt, hash.

Decrypt

The process by which a string is decompiled using a standard leading algorithm, from a long random looking hash, into a clean string or integer, utilizing the right: algorithm, salt, hash.

Salt

An unknown third variable, used for encryption/decryption. Basically it's a wildcard variable, that is vital as an encryption algorithm parameter.

Algorithms:

md5
blowfish
sha256
sha512

md5();

Using md5() you can encrypt any string into a random string hash

Example 1
Problem: Needs a salt, anyone can decrypt

```
<?php
$somestring = "Hello World!";
$encrypted  = md5($data);
```

```
echo $encrypted;
?>
```

Result:

b10a8db164e0754105b7a99be72e3fe5
32 character long string.

sha1();

Using sha1() you can encrypt any string into a random string hash

Example 2 – Static Salt
Problem: Exploitable, easy to decrypt entire table, if single salt stolen

```
<?php
$somestring = "Hello World!";
$salt = "GFSDGHRFW#qv09d0aq2vQ";

echo sha1($somestring) . "<br>";
echo sha1($salt. $somestring);
?>
```

Result:

2ef7bde608ce5404e97d5f042f95f89f1c232871
3b87fda18c85f70cd43b81b4fffb9b8f88b652c8

Example 3 - Using a unique, randomly generated salt
Note: You must store the salt value for each generated hash

```
<?php
$somestring = "Hello World!";

function generate_random(){
        return substr(sha1(mt_rand()), 0, 22);
}

$random_unique = generate_random();  /// Store value in database table with each user
$encrypted = sha1($random . $somestring);

echo $somestring . "<br>";
echo $random_unique . "<br>";
echo $encrypted;
```

```
?>
```

Result:

Hello World!
3079C2fabf848dd75a08f5
2ef7bde608ce5404e97d5f042f95f89f1c232871

Explanation:

A random number is salted using sha1, generating a hash. You must store this hash, in order to validate if a hash decrypts, with the original string of “Hello World!”

crypt():

crypt($string, $salt);

$salt begins with the algorithm we're using so for example:

md5 = $1 e,g, 1yourstring$
blowfish = $2a e.g. $2a$07$usesomesalt$
sha256 = $5 e.g. 5rounds=5000$usesomesalt$
sha512 = $6 e.g. 6rounds=5000$usesomesalt$ *recommended

$salt also contains other parameters needed for the encryption algorith such as “rounds” or “cost” for the server CPU and time. Ideally it should take the slowest time possible to decrypt, while affording your cpu performance.

Put it all together:

spinhash.php

```
<?php
class SpinHash {

   // SHA-512
   private static $algo = '$6';

   // parameter
   private static $cost = '$rounds=5000';

   // generate random sha1 salt
```

```
    public static function unique_salt() {
        return substr(sha1(mt_rand()),0,22);
    }

    // generate a hash
    public static function hash($password) {
            $salt = self::unique_salt(); // cache in database elsewhere
        return crypt($password,
                    self::$algo .
                    self::$cost .
                    '$' . $salt);
     }

    // confirm and compare a password against a cached hash
    public static function validate($hash, $password) {

            $full_salt = substr($hash, 0, 31);       //// the amount of salt visible in the string
            $new_hash = crypt($password, $full_salt);
            return ($hash == $new_hash);
    }
}
```

implementation – hash_test.php

```
<?php
require('spinhash.php');

$somestring = "Hello World!";
$encrypt = SpinHash::hash($somestring);
echo $encrypt . "<br>";

if(SpinHash::validate($encrypt, $somestring)){
            echo "true";
}else{
            echo "false";
}
?>
```

Try it

Visit http://127.0.0.1/site/hash_test.php then notice the displayed result will contain:

- a hash is encrypted with: **crypt(),** sha512, salted using a random(**mt_rand()**), generated(**sha1**) hash.

- true/ false if it validates with the string “Hello World!”