Introduction to Computer Security Module G6077

Password hashing

Learning objectives:

Use crypt method to secure password

Use password_hash() and password_verify()

Other factors like cost consideration in securing password

Use password_get_info and rehashing old hashes

Contents

Learning objectives: 1
Crypt () 3
Example-01 3
Example-02 3
Task-01 4
password_hash() 4
Example-03 4
Task-02 5
Cost in password cracking 5
Example-04 5
Example-05 6
Example-06 6
Task-03 6
password_verify() 7
Example07 7
Task-04 7
password_needs_rehash () 7
Example08 8
Password_get-info () 8

Crypt ()

Crypt() – It is a one way string hashing which creates a weak hash without the salt by default. One need to specify salt.
Syntax: crypt (string $str [, string $salt]) : string

Parameters:
str: The string to be hashed.
salt: An optional salt string to base the hashing on. If not provided, the behaviour is defined by the algorithm implementation and can lead to unexpected results.
return values: Returns the hashed string or a string that is shorter than 13 characters and is guaranteed to differ from the salt on failure.

Important: When validating passwords, a string comparison function that isn't vulnerable to timing attacks should be used to compare the output of crypt() to the previously known hash. PHP 5.6 onwards provides hash_equals() for this purpose.

Example-01

```php
<?php
// Set the password
$password = 'mypassword';
// Get the hash, letting the salt be automatically generated
$hash = crypt($password);
echo($hash);
?>
//note the output hash for a later task
```


Example-02


```php
<?php
    $hashed_password = crypt('mypassword');
  // let the salt be automatically generated

    /* You should pass the entire results of crypt() as the
    salt for comparing a   password, to avoid problems when
    different hashing algorithms are used. (As
    it says above, standard DES-based password hashing uses
    a 2-character salt,but MD5-based hashing uses 12.) */

  $user_input="mypassword";

    if (hash_equals($hashed_password, crypt($user_input, $ha
    shed_password)))
  {
       echo "Password verified!";
    }
?>

//note the output hash
```

Task-01

In your PHP application register another user. Use `crypt()` to secure the password of this new user. Verified that the user can log back in.

Logic in this task is the same as task02 in lab4-part A, where you used base64 to store an encrypted password and then verified that the user can logged back in.

Username and/or password for the new user should be the word crypt. I mentioned this before that we will be using John the ripper to try to crack these different passwords. It will help us to recognise the result of cracking.

password_hash()

`password_hash()` is a simple `crypt()` wrapper and compatible with existing password hashes. Use of `password_hash()` is encouraged. password hashes created by `crypt()` can be used with `password_hash()`.

Do you know the concept of wrapper? What does wrapper classes of primitives in Java do e.g. Integer, Double?

Syntax:

```
password_hash ( string $password , mixed $algo [, array $options ] ): string
```


Algorithms	Description
PASSWORD_DEFAULT	Use bcrypt algorithm, may change in the near future.
PASSWORD_BCRYPT	Use Blowfish algorithm to create the hash.
PASSWORD_ARGON2I	Use the Argon2i hashing algorithm to create the hash. This algorithm is only available if PHP has been compiled with Argon2 support.
PASSWORD_ARGON2ID	Use the Argon2id hashing algorithm to create the hash. This algorithm is only available if PHP has been compiled with Argon2 support.

Example-03

```php
<?php
/**
 * We just want to hash our password using the current DEFAULT algorithm.
 * This is presently BCRYPT, and will produce a 60 character result.
 *
 * Beware that DEFAULT may change over time, so you would want to prepare
 * By allowing your storage to expand past 60 characters (255 would be good)
 */
echo password_hash("IntroductionToComputer", PASSWORD_DEFAULT);
?>
```

Output: `$2y$10$.vGA1O9wmRjrwAVXD98HNOgsNpDczlqm3Jq7KnEd1rVAGv3Fykk1a`

Task-02

Using `password_hash()`, generate hashes for `mypassword` and `mysecretpassword`. Compare the first four characters of hashes with the example-03. Are the first four

characters same? If a hash value is given to you which is starting from the same four characters, what will be your first thought? Check the first three or four characters of the examples one and two.

There are predefined methods that help to identify what cipher has been used to hash an asset like password. Cracking password are quite resource consuming. Digital forensic experts often use methods like password_get_info() to plan password cracking in a away that focus on information returned by these as these methods return information like which particular cipher is used. First few characters in the hash value often indicate the cipher used.

Cost in password cracking

cost – parameter used in password hashing. A cost is a measure of how many times to run the hash -- how slow it is. You want it to be slow. Again, this is a redundant layer of security for if the hashed passwords are stolen. It makes it prohibitively expensive to brute-force anything.

If attacker got hold of the hashed values of passwords from the database. A usual mechanism is to compare it against hash value of most common passwords with salt values to find user credentials.

GPU, FPGA or ASIC can be useful in finding a match as they can run millions of hashes of attempt. In this regard, parameter cost plays a key role. A high cost value means, the amount of work (CPU time) necessary to compute the hash increases exponentially. A typical cost factor might increase the number of operations necessary to compute a password hash by a factor of 100,000 or more. The idea being that this doesn't significantly increase the cost of verifying a hash for authentic use-cases (with, say, >50% of attempts being successful), but is a dramatic penalty for someone who guesses incorrectly virtually all of the time.

The cost can range from 4 to 31. I would suggest that you use the highest cost that you can, while keeping response time reasonable.

Example-04

```php
<?php
/**
 * In this case, we want to increase the default cost for BCRYPT to
12.
 * Note that we also switched to BCRYPT, which will always be 60 cha
racters.
 */

// => use associative array concept of PHP. Examples of it were
provided in basic to PHP slides
$options = [
    'cost' => 12,
];
echo password_hash("IntroductionToComputerSecurity", PASSWORD_
BCRYPT, $options);
?>

Output: $2y$12$QjSH496pcT5CEbzjD/vtVeH03tfHKFy36d4J0Ltp3lRtee9HDxY3K
```

Note: the first four characters.

Alternative method for cost:

```
$hash = password_hash($password, PASSWORD_BCRYPT, array("cost" =>
10));
```

Example-05

```
<?php
/**
 * This code will benchmark your server to determine how high of a cost you can
 * afford. You want to set the highest cost that you can without slowing down
 * you server too much. 8-10 is a good baseline, and more is good if your servers
 * are fast enough. The code below aims for ≤ 50 milliseconds stretching time,
 * which is a good baseline for systems handling interactive logins.
 */
$timeTarget = 0.05; // 50 milliseconds

$cost = 8;
do
{
     $cost++;
     $start = microtime(true);
// microtime() returns a string in the form "msec sec", where sec is the number of
seconds, link for more detail
     password_hash("test", PASSWORD_BCRYPT, ["cost" => $cost]);
     $end = microtime(true);
}
while (($end - $start) < $timeTarget);

echo "Appropriate Cost Found: " . $cost;
?>

Output: Appropriate Cost Found: 10
```


Example-06

```
<?php
echo 'Argon2i hash: ' . password_hash('ComputerSecurity', PASSWORD_ARGON2
I);
?>
```

Output:

```
Argon2i hash:
$argon2i$v=19$m=1024,t=2,p=2$YzJBSzV4TUhkMzc3d3laeg$zqU/1IN0/AogfP4cmSJI1
vc8lpXRW9/S0sYY2i2jHT0
```

//Note: It is strongly recommended that you do not generate your own salt for this function. It will create a secure salt automatically for you if you do not specify one.

Task-03

Use the cost parameter, listed in example 05, determine a good cost for using Argon2i

password_verify()

Verifies that a password matches a hash.
Syntax: `password_verify ( string $password , string $hash ) : bool`
Paramters:
Password: the user's password
Hash: A hash created by password_hash().
Return values: TRUE if the password and hash match, or FALSE otherwise.

Example07

```
<?php
// See the password_hash() example to see where this came from.

$hash = '$2y$07$BCryptRequires22Chrcte/VlQH0piJtjXl.0t1XkA8pw9dMXTpOq';

if (password_verify('Introduction to Computer', $hash))
{
    echo 'Password is valid!';
}
else
{
    echo 'Invalid password.';
}
?>
```


Task-04

Register two users in your PHP application using BCRYPT and Argon2i. Verify that the users can log back in to the application.
User 1: Bcrypt Password: Bcrypt
User2: Argon2i Password: Argon2i

password_needs_rehash ()

password_needs_rehash — Checks if the given hash matches the given options

Syntax:
bool password_needs_rehash (string $hash , int $algo [, array $options])

Parameters:
Hash: A hash created by password_hash().
Algo: A password algorithms constants denoting the algorithm to use when hashing the password.
Options: An associative array containing options. If the cost has changed due to hardware improvement or a new hashing algorithm is available.

Example08

```php
<?php

$new = [
  'options' => ['cost' => 11],
  'algo' => PASSWORD_DEFAULT,
  'hash' => null
];

$password = 'rasmuslerdorf';

//stored hash of password
$oldHash = '$2y$07$BCryptRequires22Chrcte/VlQH0piJtjXl.0t1XkA8pw9dMXTpOq';

//verify stored hash against plain-text password
if (true === password_verify($password, $oldHash))
{
  //verify legacy password to new password_hash options

 if (true === password_needs_rehash($oldHash, $new['algo'], $new['options']))
  {
    //rehash/store plain-text password using new hash
    $newHash = password_hash($password, $new['algo'], $new['options']);
    echo $newHash;
  }
}
?>
```


Password_get-info ()

password_get_info — Returns information about the given hash

Syntax: `password_get_info ( string $hash ) : array`

print_r() displays information about a variable in a way that's readable by humans.

Example09

```php
<?php
 print_r (password_get_info (
 '$2y$07$BCryptRequires22Chrcte/VlQH0piJtjXl.0t1XkA8pw9dMXTpOq' ));
?>

Output:
Array ( [algo] => 1 [algoName] => bcrypt [options] => Array ( [cost] => 7 ) )
```