drupal / **drupal**
mirrored from http://git.drupal.org/project/drupal.git

Branch: 7.x | **drupal / includes / password.inc** | Find file | Copy path

Fetching contributors…

Cannot retrieve contributors at this time

292 lines (270 sloc) | 9.3 KB

```php
<?php

/**
 * @file
 * Secure password hashing functions for user authentication.
 *
 * Based on the Portable PHP password hashing framework.
 * @see http://www.openwall.com/phpass/
 *
 * An alternative or custom version of this password hashing API may be
 * used by setting the variable password_inc to the name of the PHP file
 * containing replacement user_hash_password(), user_check_password(), and
 * user_needs_new_hash() functions.
 */

/**
 * The standard log2 number of iterations for password stretching. This should
 * increase by 1 every Drupal version in order to counteract increases in the
 * speed and power of computers available to crack the hashes.
 */
define('DRUPAL_HASH_COUNT', 15);

/**
 * The minimum allowed log2 number of iterations for password stretching.
 */
define('DRUPAL_MIN_HASH_COUNT', 7);

/**
 * The maximum allowed log2 number of iterations for password stretching.
 */
define('DRUPAL_MAX_HASH_COUNT', 30);

/**
 * The expected (and maximum) number of characters in a hashed password.
 */
define('DRUPAL_HASH_LENGTH', 55);

/**
 * Returns a string for mapping an int to the corresponding base 64 character.
 */
function _password_itoa64() {
  return './0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz';
}

/**
 * Encodes bytes into printable base 64 using the *nix standard from crypt().
 *
 * @param $input
 *   The string containing bytes to encode.
 * @param $count
 *   The number of characters (bytes) to encode.
 *
 * @return
 *   Encoded string
 */
function _password_base64_encode($input, $count) {
  $output = '';
  $i = 0;
  $itoa64 = _password_itoa64();
  do {
    $value = ord($input[$i++]);
    $output .= $itoa64[$value & 0x3f];
    if ($i < $count) {
      $value |= ord($input[$i]) << 8;
    }
    $output .= $itoa64[($value >> 6) & 0x3f];
    if ($i++ >= $count) {
      break;
    }
    if ($i < $count) {
      $value |= ord($input[$i]) << 16;
    }
    $output .= $itoa64[($value >> 12) & 0x3f];
    if ($i++ >= $count) {
      break;
    }
    $output .= $itoa64[($value >> 18) & 0x3f];
  } while ($i < $count);

  return $output;
}

/**
 * Generates a random base 64-encoded salt prefixed with settings for the hash.
 *
 * Proper use of salts may defeat a number of attacks, including:
 *  - The ability to try candidate passwords against multiple hashes at once.
 *  - The ability to use pre-hashed lists of candidate passwords.
 *  - The ability to determine whether two users have the same (or different)
 *    password without actually having to guess one of the passwords.
 *
 * @param $count_log2
 *   Integer that determines the number of iterations used in the hashing
 *   process. A larger value is more secure, but takes more time to complete.
 *
 * @return
 *   A 12 character string containing the iteration count and a random salt.
 */
function _password_generate_salt($count_log2) {
  $output = '$S$';
  // Ensure that $count_log2 is within set bounds.
  $count_log2 = _password_enforce_log2_boundaries($count_log2);
  // We encode the final log2 iteration count in base 64.
  $itoa64 = _password_itoa64();
  $output .= $itoa64[$count_log2];
  // 6 bytes is the standard salt for a portable phpass hash.
  $output .= _password_base64_encode(drupal_random_bytes(6), 6);
  return $output;
}

/**
 * Ensures that $count_log2 is within set bounds.
 *
 * @param $count_log2
 *   Integer that determines the number of iterations used in the hashing
 *   process. A larger value is more secure, but takes more time to complete.
 *
 * @return
 *   Integer within set bounds that is closest to $count_log2.
 */
function _password_enforce_log2_boundaries($count_log2) {
  if ($count_log2 < DRUPAL_MIN_HASH_COUNT) {
    return DRUPAL_MIN_HASH_COUNT;
  }
  elseif ($count_log2 > DRUPAL_MAX_HASH_COUNT) {
    return DRUPAL_MAX_HASH_COUNT;
  }

  return (int) $count_log2;
}

/**
 * Hash a password using a secure stretched hash.
 *
 * By using a salt and repeated hashing the password is "stretched". Its
 * security is increased because it becomes much more computationally costly
 * for an attacker to try to break the hash by brute-force computation of the
 * hashes of a large number of plain-text words or strings to find a match.
 *
 * @param $algo
 *   The string name of a hashing algorithm usable by hash(), like 'sha256'.
 * @param $password
 *   Plain-text password up to 512 bytes (128 to 512 UTF-8 characters) to hash.
 * @param $setting
 *   An existing hash or the output of _password_generate_salt().  Must be
 *   at least 12 characters (the settings and salt).
 *
 * @return
 *   A string containing the hashed password (and salt) or FALSE on failure.
 *   The return string will be truncated at DRUPAL_HASH_LENGTH characters max.
 */
function _password_crypt($algo, $password, $setting) {
  // Prevent DoS attacks by refusing to hash large passwords.
  if (strlen($password) > 512) {
    return FALSE;
  }
  // The first 12 characters of an existing hash are its setting string.
  $setting = substr($setting, 0, 12);

  if ($setting[0] != '$' || $setting[2] != '$') {
    return FALSE;
  }
  $count_log2 = _password_get_count_log2($setting);
  // Hashes may be imported from elsewhere, so we allow != DRUPAL_HASH_COUNT
  if ($count_log2 < DRUPAL_MIN_HASH_COUNT || $count_log2 > DRUPAL_MAX_HASH_COUNT) {
    return FALSE;
  }
  $salt = substr($setting, 4, 8);
  // Hashes must have an 8 character salt.
  if (strlen($salt) != 8) {
    return FALSE;
  }

  // Convert the base 2 logarithm into an integer.
  $count = 1 << $count_log2;

  // We rely on the hash() function being available in PHP 5.2+.
  $hash = hash($algo, $salt . $password, TRUE);
  do {
    $hash = hash($algo, $hash . $password, TRUE);
  } while (--$count);

  $len = strlen($hash);
  $output =  $setting . _password_base64_encode($hash, $len);
  // _password_base64_encode() of a 16 byte MD5 will always be 22 characters.
  // _password_base64_encode() of a 64 byte sha512 will always be 86 characters.
  $expected = 12 + ceil((8 * $len) / 6);
  return (strlen($output) == $expected) ? substr($output, 0, DRUPAL_HASH_LENGTH) : FALSE;
}

/**
 * Parse the log2 iteration count from a stored hash or setting string.
 */
function _password_get_count_log2($setting) {
  $itoa64 = _password_itoa64();
  return strpos($itoa64, $setting[3]);
}

/**
 * Hash a password using a secure hash.
 *
 * @param $password
 *   A plain-text password.
 * @param $count_log2
 *   Optional integer to specify the iteration count. Generally used only during
 *   mass operations where a value less than the default is needed for speed.
 *
 * @return
 *   A string containing the hashed password (and a salt), or FALSE on failure.
 */
function user_hash_password($password, $count_log2 = 0) {
  if (empty($count_log2)) {
    // Use the standard iteration count.
    $count_log2 = variable_get('password_count_log2', DRUPAL_HASH_COUNT);
  }
  return _password_crypt('sha512', $password, _password_generate_salt($count_log2));
}

/**
 * Check whether a plain text password matches a stored hashed password.
 *
 * Alternative implementations of this function may use other data in the
 * $account object, for example the uid to look up the hash in a custom table
 * or remote database.
 *
 * @param $password
 *   A plain-text password
 * @param $account
 *   A user object with at least the fields from the {users} table.
 *
 * @return
 *   TRUE or FALSE.
 */
function user_check_password($password, $account) {
  if (substr($account->pass, 0, 2) == 'U$') {
    // This may be an updated password from user_update_7000(). Such hashes
    // have 'U' added as the first character and need an extra md5().
    $stored_hash = substr($account->pass, 1);
    $password = md5($password);
  }
  else {
    $stored_hash = $account->pass;
  }

  $type = substr($stored_hash, 0, 3);
  switch ($type) {
    case '$S$':
      // A normal Drupal 7 password using sha512.
      $hash = _password_crypt('sha512', $password, $stored_hash);
      break;
    case '$H$':
      // phpBB3 uses "$H$" for the same thing as "$P$".
    case '$P$':
      // A phpass password generated using md5.  This is an
      // imported password or from an earlier Drupal version.
      $hash = _password_crypt('md5', $password, $stored_hash);
      break;
    default:
      return FALSE;
  }
  return ($hash && $stored_hash == $hash);
}

/**
 * Check whether a user's hashed password needs to be replaced with a new hash.
 *
 * This is typically called during the login process when the plain text
 * password is available. A new hash is needed when the desired iteration count
 * has changed through a change in the variable password_count_log2 or
 * DRUPAL_HASH_COUNT or if the user's password hash was generated in an update
 * like user_update_7000().
 *
 * Alternative implementations of this function might use other criteria based
 * on the fields in $account.
 *
 * @param $account
 *   A user object with at least the fields from the {users} table.
 *
 * @return
 *   TRUE or FALSE.
 */
function user_needs_new_hash($account) {
  // Check whether this was an updated password.
  if ((substr($account->pass, 0, 3) != '$S$') || (strlen($account->pass) != DRUPAL_HASH_LENGTH)) {
    return TRUE;
  }
  // Ensure that $count_log2 is within set bounds.
  $count_log2 = _password_enforce_log2_boundaries(variable_get('password_count_log2', DRUPAL_HASH_COUNT));
  // Check whether the iteration count used differs from the standard number.
  return (_password_get_count_log2($account->pass) !== $count_log2);
}
```