If the user ID and password are externally stored in a file or database, the information will also travel outside the program. The program will no longer have control over the security of these items once they reside in the file or database. This could allow hackers access to the information. Security, as mentioned, has to be a team effort among the programmer, data administrator, and network administrator.

It is common practice to encrypt the password to reduce the chance that hackers will discover the authentication information (or any other secure information). Many PHP books demonstrate the use of the MD5 hash technique. However, over the last several years vulnerabilities have been discovered in this encyption style.

PHP 5.5 included the method `password_hash`, which will be adjusted over time to use the most secure encyption hashing techniques avaliable.

Programming note—Caution should be used when storing the encrypted version of the password. The size of the resulting encryption will increase with new hash versions. A size of 255 characters is likely to be large enough for the many years. The number of milliseconds needed for this hash increases with the size and type of the encryption. Advanced programmers may want to do some testing on their servers for time costs.

Visit http://php.net/manual/en/function.password-hash.php for more information.

Programming note—You cannot do a simple comparison with the hashed password created by PHP's `password_hash` method. The hash produced includes the encryption type, a salt value, and the hashed password.

You only need to replace one line of code in the example to verify the password. You can replace

```
$valid = ((in_array($userid, $valid_userids)) && ($password == $valid_useridpasswords[$userid]));
```

with

```
$valid =( (in_array($userid, $valid_userids)) && (password_verify($password,
$valid_useridpasswords[$userid]));
```

If you placed the encypted password in the `$valid_useridpasswords` array, the validation technique would not require any other changes. The PHP `password_verify` method will encrpt the password provided by the user and compare it to the existing encyrpted password. If they match, it will return `TRUE`.

When using XML or JSON files, you can use the same logic used in the constructor for the `dogdata.php` program from Chapter 6, to retrieve the valid user ID and password information. The only changes needed are to the `if` statement, which determines the location of the user ID and password file, and to the last line in the constructor to place the array produced to `$valid_useridpasswords` instead of `dogs_array`.

```
<users>
<user>
<userid>Fredfred</userid>
<password>$2y$10$VosI32FejL.bOMaCjGbBp.Jre6Ipa.tLYQrVqj9kiVpef5zZ25qQK</password>
</user>
<user>
<userid>Petepete</userid>
<password>$2y$10$FdbXxIVXmVOHtaBNxB8vzupRBJFCqUyOTJXrlpNdrLOHKQ/U.jFHO</password>
</user>
</users>
```