```
<user>
<userid>Poppoppop</userid>
<password>$2y$10$C1jXhTlOmyamuLKhZxK5m.4X4TVcdeFbeLSBIA7l4fx6tUnC8vrg6</password>
<datestamp>2015-06-04</datestamp>
<attempts>1</attempts>
<lastattempt>08062015113200</lastattempt>
<validattempt>08062015113038</validattempt>
</user>
</users>
```

The `$newupstring` can be adjusted in the `registration.php` program (Example 7-3) to add the authentication fields.

```
$newupstring = "<user>\n<userid>" . $userid . "</userid>\n<password>" . $hashed_password .
"</password>\n";
$newupstring .= "<datestamp>" . date('Y-m-d', strtotime('+30 days')) . "</datestamp>\n";
$newupstring .= "<attempts>0</attempts>\n<lastattempt>" . date('mdYhis') . "</lastattempt>\n";
$newupstring .= "<validattempt>" . date('mdYhis') . "</validattempt>\n</user>\n</users>";
```

The PHP method `strtotime` will parse any standard date and time format and attempt to convert it to the UNIX date time format (which PHP uses). In this example, the method provides the ability to add 30 days to the current date, which in turn will be used to determine if a password has expired. Alternatively, an expired date (such as the day before the current date) could be placed in this field when user IDs are created in bulk (such as populating student IDs in a course management system). This would force the users to change the password the first time they sign in to the system. The expire date is stored in `datestamp` for use when the user logs in to the system.

The `attempts` tag, in the example, will record how many times the user tries to sign in with a bad user ID password combination (reset to zero when a valid login occurs). If the date and time in `lastattempt` are in five minutes, and the value of attempts is 3 or greater, the user must wait until more than five minutes has expired since the last attempt to log in. As with most login systems, even if the user logs in with valid information, the last invalid login must be five or more minutes ago. The last valid login date and time is also recorded in the `validattempt` tags. Although this is not used for authentication in this example, it is important to keep track of all valid logins.

To keep the code as simple as possible in the main section of the program, the code that looks up the location of the user ID and password file has been moved to the method `retrieve_useridpasswordfile`. Saving the data in the XML file has also been moved to the method `saveupfile`. No changes (except for the addition of more XML tags as mentioned previously) have occurred in the code for these methods.

Example 7-4. The login.php file with password timeout and three tries timeout

```
<?php
session_start();
$user_log_file = "user.log";
$passed = FALSE;
function saveupfile($dog_data_xml,$valid_useridpasswords)
{
$xmlstring = '<?xml version="1.0" encoding="UTF-8"?>';
           $xmlstring .= "\n<users>\n";
         foreach($valid_useridpasswords as $users)
        {
        foreach($users as $user)
```