



Join InMotion Hosting for \$3.49/mo & get a year on Tuts+ FREE (worth \$180).
[Start today](#)

[Dismiss](#) ✕



Code



Categories

Categories

PHP

A Better Login System

by [Andrew Steenbuck](#) 26 Mar 2009 [269 Comments](#)



18



5



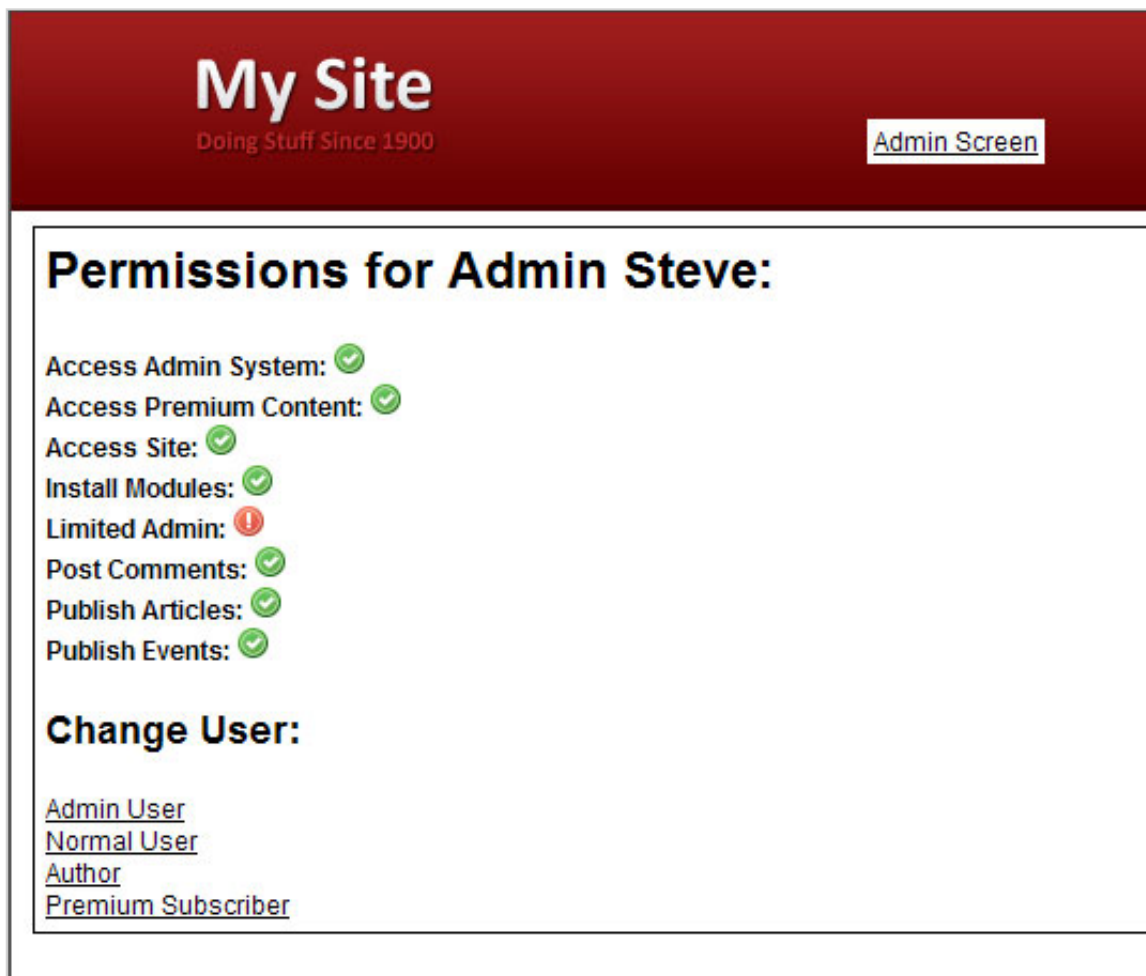
14



Net.tuts+ has [published several great tutorials](#) on user login systems. Most tutorials only deal with authenticating the user, which allows for two levels of security: logged in and not logged in. For many sites, a finer degree of control is needed to control where users can go and what they can do. Creating an access control list (ACL) system will give you the flexibility for granular permissions.



Introduction



Imagine you are running a great tutorial site that lets users learn about a wide variety of web development techniques. In addition to your normal readers, you have some premium subscription members, as well contributing authors and administrators.

Your problem

You want to restrict users' to only specific pages that their particular account allows access to.

The solution

Implementing an access control list will allow you a great deal of control over what users can and cannot access on your site.

If you view the demo, available with the downloadable source code, you will be greeted with an index page that tests the ACL for each user. You can select different links at the bottom to view the ACL for the different users. If you click on the 'Admin Screen' link near the top, you can view a sample of the admin interface that allows you to manage the users, roles, and permissions. NOTE: The admin system will perform a database restore every 30 minutes to make sure everything stays on the up and up. The download files also implement the ACL security on the admin site, so

if user number one doesn't have the 'access admin' permission, you won't be able to access the admin site.

This system will enable you to create different groups of users (i.e. guests, premium members, contributors, and admins). We will be able to set unique permissions for each group, as well as for individual users. Let's get started by setting up our MySQL database.

Step 1: Create the Database

Our ACL will be stored in a relational database using six tables (including the table for users). You should already have a database set up in your host environment. We will create the following table structure:

	Table	Action						Records💡	Type
<input type="checkbox"/>	permissions							8	InnoDB
<input type="checkbox"/>	roles							4	InnoDB
<input type="checkbox"/>	role_perms							15	InnoDB
<input type="checkbox"/>	users							4	InnoDB
<input type="checkbox"/>	user_perms							0	InnoDB
<input type="checkbox"/>	user_roles							7	InnoDB
6 table(s)		Sum						38	InnoDB
	Check All / Uncheck All						With selected:		

The code to create the database is available in the source files (install.sql), and there is also another file (sampleData.sql) that will create 4 sample users, along with several roles and permissions for you to test with. Simply open the files with your favorite text editor, and copy/paste the code into the SQL panel in phpMyAdmin.

Step 2: Database Include

We need to create an include file so that we may connect to our database. Create a file called assets/php/database.php and add the following code to it (replace the variable values with the information appropriate for your hosting situation):

```
01 <?php
02 session_start();
03 ob_start();
04 $hasDB = false;
05 $server = 'localhost';
06 $user = 'root';
07 $pass = 'mysql';
08 $db = 'acl_test';
09 $link = mysql_connect($server,$user,$pass);
10 if (!is_resource($link)) {
11     $hasDB = false;
12     die("Could not connect to the MySQL server at localhost.");
13 } else {
14     $hasDB = true;
15     mysql_select_db($db);
16 }
17 ?>
```

On the first line of the code, we call **session_start()**; we will not actually use the session variables but you will need it as part of the user login system. Then, we call **ob_start()** to create an output buffer. Typically, when PHP generates the page, it is sent to the browser as it is generating. By using **ob_start()**, the page and headers aren't sent to the browser until they've loaded completely, or until we call **ob_end_flush()**. By buffering the page, we are able to redirect using PHP at any point on the page, instead of just at the top. After the headers are sent, our only redirect option is with JavaScript. An enterprising hacker could easily turn JavaScript off, and then see our unsecured page in all it's glory. This one line allows us to deny the user access at any point in the page if needed.

Lines 4-8 set up our variables. **\$hasDB** is a boolean used to determine if we are connected. **\$server**, **\$user**, **\$pass**, and **\$db** are the connection arguments for the server. Line 9 connects to the server, while line 10 determines if the connection was successful. If it was, we select the database to use; if it wasn't, we display an error message using **die()**.

Step 3: Create the ACL Class

This step is fairly long, as we are creating the ACL class that will form the basis of our system. I apologize in advance for the length of this step.

```

class ACL
{
    var $perms = array();           //Array : Stores the permissions for the user
    var $userID = 0;                //Integer : Stores the ID of the current user
    var $userRoles = array();       //Array : Stores the roles of the current user

    function __constructor($userID = '')
    {
        if ($userID != '')
        {
            $this->userID = floatval($userID);
        } else {
            $this->userID = floatval($_SESSION['userID']);
        }
        $this->userRoles = $this->getUserRoles('ids');
        $this->buildACL();
    }

    function ACL($userID = '')
    {
        $this->__constructor($userID);
        //crutch for PHP4 setups
    }

    function buildACL()
    {
        //first, get the rules for the user's role
        if (count($this->userRoles) > 0)
        {
            $this->perms = array_merge($this->perms, $this->getRolePerms($this->userRoles));
        }
        //then, get the individual user permissions
        $this->perms = array_merge($this->perms, $this->getUserPerms($this->userID));
    }
}

```

Our ACL system will be object-oriented, so let's start creating the class file. We start by adding the class definition, variable definitions, and the constructor to the file `/assets/php/class.acl.php`:

```

01 <?php
02 class ACL
03 {
04     var $perms = array();           //Array : Stores the permissions
05     var $userID = 0;                //Integer : Stores the ID of the
06     var $userRoles = array();       //Array : Stores the roles of th
07
08     function __constructor($userID = '')
09     {
10         if ($userID != '')
11         {
12             $this->userID = floatval($userID);
13         } else {
14             $this->userID = floatval($_SESSION['userID']);
15         }
16         $this->userRoles = $this->getUserRoles('ids');
17         $this->buildACL();
18     }
19     function ACL($userID='')
20     {
21         $this->__constructor($userID);
22     }

```

Analysis

After creating the class definition, we create the three class variables to store the information that will be used in the generation of the ACL.

The Constructor Method

The **__constructor()** function is used to initialize the object when we want to load an ACL. It is called automatically when we call **new ACL()**; It is then passed a single, optional argument of the user to load the ACL for. Inside the constructor, we check to see if a user ID was passed in. If no ID was passed, we assume that we will load the ACL for the currently logged in user; so we read in the session variable for that. Alternatively, if we pass in a user ID, it allows us to read and edit the ACL for a user other than the one logged in (useful for your admin page).

After we read in the user ID, we call **getUserRoles()** to generate an array of the roles the user is assigned to and store it in the **\$userRoles** class variable. At the end of the constructor, we call **buildACL()** to generate the actual ACL. The method named **ACL()** is a crutch for PHP4 installs. When you call **new ACL()** in PHP5, the PHP interpreter runs the **__constructor()** method. However, when you run the same code in PHP4, the interpreter runs **ACL()**. By providing a method named the same as the class, we make the class PHP4 compatible.

Any time we create a new ACL object by passing in a user ID, that object will hold the permissions for the user who was passed in.

Helper Methods

Now, lets add some more helper methods to the same class file. These methods will provide support to the other methods by performing specialized tasks:

```

01  function getUserRoles()
02  {
03      $strSQL = "SELECT * FROM `user_roles` WHERE `userID` = " . fl
04      $data = mysql_query($strSQL);
05      $resp = array();
06      while($row = mysql_fetch_array($data))
07      {
08          $resp[] = $row['roleID'];
09      }
10      return $resp;
11  }
12  function getAllRoles($format='ids')
13  {
14      $format = strtolower($format);
15  }
```

```

16 $strSQL = "SELECT * FROM `roles` ORDER BY `roleName` ASC";
17 $data = mysql_query($strSQL);
18 $resp = array();
19 while($row = mysql_fetch_array($data))
20 {
21     if ($format == 'full')
22     {
23         $resp[] = array("ID" => $row['ID'], "Name" => $row['ro
24     } else {
25         $resp[] = $row['ID'];
26     }
27 }
28 return $resp;
29 }
30 function buildACL()
31 {
32     //first, get the rules for the user's role
33     if (count($this->userRoles) > 0)
34     {
35         $this->perms = array_merge($this->perms, $this->getRolePer
36     }
37     //then, get the individual user permissions
38     $this->perms = array_merge($this->perms, $this->getUserPerms($
39 }
40 function getPermKeyFromID($permID)
41 {
42     $strSQL = "SELECT `permKey` FROM `permissions` WHERE `ID` = "
43     $data = mysql_query($strSQL);
44     $row = mysql_fetch_array($data);
45     return $row[0];
46 }
47 function getPermNameFromID($permID)
48 {
49     $strSQL = "SELECT `permName` FROM `permissions` WHERE `ID` =
50     $data = mysql_query($strSQL);
51     $row = mysql_fetch_array($data);
52     return $row[0];
53 }
54 function getRoleNameFromID($roleID)
55 {
56     $strSQL = "SELECT `roleName` FROM `roles` WHERE `ID` = " . fl
57     $data = mysql_query($strSQL);
58     $row = mysql_fetch_array($data);
59     return $row[0];
60 }
61 function getUsername($userID)
62 {
63     $strSQL = "SELECT `username` FROM `users` WHERE `ID` = " . fl
64     $data = mysql_query($strSQL);
65     $row = mysql_fetch_array($data);
66     return $row[0];
67 }

```

getUserRoles()

getUserRoles() will return an array of roles the current user it is assigned to. First, we will build the appropriate SQL statement and execute it. Using **while()**, we loop

through all of the matching results, and finally return an array of the IDs. Likewise, **getAllRoles()** will return all of the available roles (not just the ones the user is assigned to). Based on the value of the argument, **\$format**, it will return an array of IDs for all the roles, or an array of associative array with the ID and name of each role. This allows our function to do double duty. If we want to use the array of user roles in MySQL, we need an array of role IDs; but if we want to display the roles on our page, it would be helpful to have one array with all the info in it.

buildACL

buildACL() generates the permissions array for the user, and is the heart of the system. First, we check to see if the user is assigned to any roles. If they are, we use **array_merge()** to combine the existing permissions array with the new array returned from the call to **getRolePerms()** (which gets all the permissions for all the roles the user is assigned to). Then we do the same for the individual user permissions, this time calling **getUserPerms()**. It is important that we read the user perms second because **array_merge()** overwrites duplicate keys. Reading the user permissions second ensures that the individual permissions will override any permissions inherited from the user's roles.

All of the functions **getPermKeyFromID()**, **getPermNameFromID()**, **getRoleNameFromID()** and **getUsername()** are simply "lookup" functions. They allow us to pass in an ID and return the appropriate text value. You can see that we build the SQL statement, then execute it and return the result. Next we will add in the two functions which will pull the permissions from the database.

```

01 function getRolePerms($role)
02 {
03     if (is_array($role))
04     {
05         $roleSQL = "SELECT * FROM `role_perms` WHERE `roleID` IN
06     } else {
07         $roleSQL = "SELECT * FROM `role_perms` WHERE `roleID` = "
08     }
09     $data = mysql_query($roleSQL);
10     $perms = array();
11     while($row = mysql_fetch_assoc($data))
12     {
13         $pK = strtolower($this->getPermKeyFromID($row['permID']));
14         if ($pK == '') { continue; }
15         if ($row['value'] === '1') {
16             $hP = true;
17         } else {
18             $hP = false;
19         }
20         $perms[$pK] = array('perm' => $pK, 'inherited' => true, 'v

```



```

21     }
22     return $perms;
23 }
24
25 function getUserPerms($userID)
26 {
27     $strSQL = "SELECT * FROM `user_perms` WHERE `userID` = " . $userID . " ";
28     $data = mysql_query($strSQL);
29     $perms = array();
30     while($row = mysql_fetch_assoc($data))
31     {
32         $pK = strtolower($this->getPermKeyFromID($row['permID']));
33         if ($pK == '') { continue; }
34         if ($row['value'] == '1') {
35             $hP = true;
36         } else {
37             $hP = false;
38         }
39         $perms[$pK] = array('perm' => $pK, 'inherited' => false, '
40     }
41     return $perms;
42 }
43 function getAllPerms($format='ids')
44 {
45     $format = strtolower($format);
46     $strSQL = "SELECT * FROM `permissions` ORDER BY `permName` AS
47     $data = mysql_query($strSQL);
48     $resp = array();
49     while($row = mysql_fetch_assoc($data))
50     {
51         if ($format == 'full')
52         {
53             $resp[$row['permKey']] = array('ID' => $row['ID'], 'N
54         } else {
55             $resp[] = $row['ID'];
56         }
57     }
58     return $resp;
59 }

```

These functions are essentially identical except for the tables they pull from. The single argument is the ID for the roles/users you want to pull. The roles function can be passed an array or an integer, while the user function can only be passed an integer. By using `is_array()`, we determine how to treat the argument for the role permission function. If it is an array, we use `implode()` to create a comma-separated-list. In either case, we use that value in the SQL. Then, we create a new empty array called `$perms` - this will store the permissions locally in the function.

Inside the `while()` loop, we perform several functions. First we generate the variable `$pK`, which we will use as the name of the array key. Because we will be looking for this value to determine if the user has a specific permission, it is important that we have it in a uniform format, which is why we are using `strtolower()`. If the key value

is blank, we skip to the next iteration using **continue**;. Next, we look at **\$row['value']** to set an implicit boolean value for the permission. This ensures that only an actual value of '1' in the table will equate with true (i.e. the user has the permission), and is important for security. Otherwise we set the permission to false. At the end of the function, we create an array with several named keys so we can get all of the information about a permission. That array is assign to a new named key in the **\$perms** array we created earlier. Note that we use **\$pK** to create an appropriately named index. Finally we return the array.

You can see that in the returned array, there is an index name 'inherited'. This has a special significance for the ACL. If a user receives a permission because it belongs to a role the user is assigned to, it is said to be inherited. If the permissions is assigned to the user manually, it is not inherited.

In **getAllPerms()**, we build a list of all available permissions. Similar to **getAllRoles()** we can pass in a format argument to determine how the results will be returned. Now for the last part of the class:

```

01     function userHasRole($roleID)
02     {
03         foreach($this->userRoles as $k => $v)
04         {
05             if (floatval($v) === floatval($roleID))
06             {
07                 return true;
08             }
09         }
10         return false;
11     }
12
13     function hasPermission($permKey)
14     {
15         $permKey = strtolower($permKey);
16         if (array_key_exists($permKey,$this->perms))
17         {
18             if ($this->perms[$permKey]['value'] === '1' || $this->
19             {
20                 return true;
21             } else {
22                 return false;
23             }
24         } else {
25             return false;
26         }
27     }
28 }
29 ?>

```

These last two methods are very important for the functionality of the ACL.

userHasRole() accepts the single argument of a role ID. By looping through all the elements in the **\$userRoles** array, we can determine if the user is assigned to that role. If they are, we return true, or false otherwise. **hasPermission()** is the method we use to determine if a user can access something. We pass in the key for the permission we want to check. We make it uniform by converting it to lowercase, and see if there is an index with that name in the **\$perms** array. If there is, we check to make sure that it is set to '1' and return true, or return false otherwise. This is the function we will use if we want to figure out if a user can do something.

Step 4: User Admin

The first part of our admin section will deal with managing users. We need to create four different interfaces to deal with the aspects of managing users: List the users so we can select one to edit, viewing a detail user listing, assign users to roles, and grant users permissions.



Open `/admin/users.php` and add the following code:

```
01 <?php
02 include("../assets/php/database.php");
03 include("../assets/php/class.acl.php");
04 $myACL = new ACL();
05
```

```

06  if ($myACL->hasPermission('access_admin') != true)<br />{<br />
07  ?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
08  <html xmlns="http://www.w3.org/1999/xhtml">
09  <head>
10  <meta http-equiv="Content-Type" content="text/html; charset=utf-8
11  <title>ACL Test</title>
12  <link href="../assets/css/styles.css" rel="stylesheet" type="text
13  </head>
14  <body>
15  <div id="header"></div>
16  <div id="adminButton"><a href="..">Main Screen</a> | <a href="in
17  <div id="page">
18      <!-- PAGE CONTENT -->
19  </div>
20  </body>
    </html>

```

As always, we need to include our database and ACL files, and set up the ACL object. Then we set up the security for the page. In this case, we are ensuring that the user has the permission 'access_admin'. If they don't, they are redirected.

NOTE: If you change the ACL permissions so that user #1 no longer has the 'access_admin' permission, you won't be able to access the admin site. Also, you must first go to /index.php before you go to any of the admin pages, as index.php sets the session variable assigning you userID #1.

Right now this is just the basic layout of the page. In the next steps, we will replace <!-- PAGE CONTENT --> above with some code to manage the users. We will use the querystring variable **\$action** to determine which of the user interfaces we should display. There are four possible values that we will address: If it is null, we display a list of the current users. If it is set to 'user', we display the form for a single user. If it is set to 'roles', we display the form to assign a user. If it is set to 'perms', we display the form to give the user permissions.

List Users

Add this code inside the div with the id 'page':

```

01  <? if ($_GET['action'] == '' ) { ?>
02      <h2>Select a User to Manage:</h2>
03      <?
04      $strSQL = "SELECT * FROM `users` ORDER BY `Username` ASC";
05      $data = mysql_query($strSQL);
06      while ($row = mysql_fetch_assoc($data))
07      {
08          echo "<a href=\"?action=user&userID=" . $row['ID'] . "\">
09      }

```

```
10 | } ?>
```

The concept here is pretty simple. We build a SQL query, run it and loop through the results. For each user, we generate a link that will enable us to edit that particular user.

Edit Individual User

Now, add this code directly under the previous code block:

```
01 <?
02 if ($_GET['action'] == 'user' ) {
03     $userACL = new ACL($_GET['userID']);
04     ?>
05     <h2>Managing <?= $myACL->getUsername($_GET['userID']); ?>:</h
06     ... Some form to edit user info ...
07     <h3>Roles for user:    (<a href="users.php?action=roles&userID
08     <ul>
09     <? $roles = $userACL->getUserRoles();
10     foreach ($roles as $k => $v)
11     {
12         echo "<li>" . $userACL->getRoleNameFromID($v) . "</li>";
13     }
14     ?>
15     </ul>
16     <h3>Permissions for user:    (<a href="users.php?action=perms&
17     <ul>
18     <? $perms = $userACL->perms;
19     foreach ($perms as $k => $v)
20     {
21         if ($v['value'] === false) { continue; }
22         echo "<li>" . $v['Name'];
23         if ($v['inherited']) { echo " (inherited)"; }
24         echo "</li>";
25     }
26     ?>
27     </ul>
28     <? } ?>
```

When we edit a user, we need to load the ACL for that user. This will enable us to see which roles and permissions they have. We start that by creating a new ACL object, and passing in the **\$userID** from the querystring (this way we load that user's ACL, instead of the logged in user). After that is where your normal edit user form would go. Typical things would be text fields to edit username, password, etc. Below that we list the roles the user is assigned to, and also provide a link so we can assign the user to other roles. Lines 10-16 load all the roles that the user is assigned to, and prints them out as list items using **foreach()**. Then we list out the user's permissions in a similar fashion. We only print out the permissions that the user has,

not ones that are set to false.

My Site
Doing Stuff Since 1900

[Main Screen](#) | [Admin Home](#)

Managing Penelope Premium:

[A complete form to edit the user info - password, email, etc - would go here.]

Roles for user: ([Manage Roles](#))

- All Users
- Premium Members

Permissions for user: ([Manage Permissions](#))

- Access Site (inherited)
- Access Premium Content (inherited)
- Post Comments (inherited)

Assign Roles

Our assign roles form will end up looking like this:

My Site
Doing Stuff Since 1900

[Main Screen](#) | [Admin Home](#)

Manage User Roles: (Author Aaron)

	Member	Not Member
Administrators	<input type="radio"/>	<input checked="" type="radio"/>
All Users	<input checked="" type="radio"/>	<input type="radio"/>
Authors	<input checked="" type="radio"/>	<input type="radio"/>
Premium Members	<input type="radio"/>	<input checked="" type="radio"/>

Submit

Cancel

Add this code right below the previous code block:

```

01  <? if ($_GET['action'] == 'roles') { ?>
02  <h2>Manage User Roles: (<?= $myACL->getUsername($_GET['userID'])
03  <form action="users.php" method="post">
04  <table border="0" cellpadding="5" cellspacing="0">
05  <tr><th></th><th>Member</th><th>Not Member</th></tr>
06  <?
07  $roleACL = new ACL($_GET['userID']);
08  $roles = $roleACL->getAllRoles('full');
09  foreach ($roles as $k => $v)
10  {
11      echo "<tr><td><label>" . $v['Name'] . "</label></td>";
12      echo "<td><input type=\"radio\" name=\"role_\" . $v['ID']
13      if ($roleACL->userHasRole($v['ID'])) { echo " checked=\"c
14      echo " /></td>";
15      echo "<td><input type=\"radio\" name=\"role_\" . $v['ID']
16      if (!$roleACL->userHasRole($v['ID'])) { echo " checked=\"
17      echo " /></td>";
18      echo "</tr>";
19  }
20  ?>
21  </table>
22  <input type="hidden" name="action" value="saveRoles" />
23  <input type="hidden" name="userID" value="<?= $_GET['userID']
24  <input type="submit" name="Submit" value="Submit" />
25  </form>
26  <form action="users.php" method="post">
27  <input type="button" name="Cancel" onclick="window.location='
28  </form>
29  <? } ?>

```

The first thing we have to do here is create a form and a table. The table will have 3 columns: one for the role, one for the member checkbox, and one for the non-member checkbox. After creating a new ACL object, we load an array of all the roles using **getAllRoles()**. That will allow us to display input elements for every role, not just the ones a user is assigned to.

Inside the **foreach()** loop, we do the following: We start a new row and print out a label with the name of the role. Then we print out a radio button input. The name and id of the radio buttons is made unique for each role by using the format "role_[roleID]" (i.e. role_0012). Lines 13 and 16 determine which of the radio buttons should be checked. The first one will be checked if the user is already assigned to the group, while the second one will be checked if they are not. Notice that one has a value of '1' (for assign), and the other has a value of '0' (for don't assign). Then we end the row.

After all that, we add in some hidden elements that tell us what we are saving, and

what user ID to save. Then we add a submit and cancel button.

Assign Permissions

The assign permissions form is similar to the roles form, but with different inputs, so let's add this code:

```

01  <? if ( $_GET['action'] == 'perms' ) { ?>
02      <h2>Manage User Permissions: (= $myACL-&gt;getUsername($_GET['
03      &lt;form action="users.php" method="post"&gt;
04          &lt;table border="0" cellpadding="5" cellspacing="0"&gt;
05              &lt;tr&gt;&lt;th&gt;&lt;/th&gt;&lt;th&gt;&lt;/th&gt;&lt;/tr&gt;
06              &lt;?
07              $userACL = new ACL($_GET['userID']);
08              $rPerms = $userACL-&gt;perms;
09              $aPerms = $userACL-&gt;getAllPerms('full');
10              foreach ( $aPerms as $k =&gt; $v )
11              {
12                  echo "&lt;tr&gt;&lt;td&gt;" . $v['Name'] . "&lt;/td&gt;";
13                  echo "&lt;td&gt;&lt;select name=\"perm_\" . $v['ID'] . \"&gt;";
14                  echo "&lt;option value=\"1\"&gt;";
15                  if ( $rPerms[$v['Key']]['value'] === true &amp;&amp; $rPerms[$
16                  echo "&gt;Allow&lt;/option&gt;";
17                  echo "&lt;option value=\"0\"&gt;";
18                  if ( $rPerms[$v['Key']]['value'] === false &amp;&amp; $rPerms[
19                  echo "&gt;Deny&lt;/option&gt;";
20                  echo "&lt;option value=\"x\"&gt;";
21                  if ( $rPerms[$v['Key']]['inherited'] == true || !arra
22                  {
23                      echo " selected=\"selected\"&gt;";
24                      if ( $rPerms[$v['Key']]['value'] === true )
25                      {
26                          $iVal = '(Allow)';
27                      } else {
28                          $iVal = '(Deny)';
29                      }
30                  }
31                  echo "&gt;Inherit $iVal&lt;/option&gt;";
32                  echo "&lt;/select&gt;&lt;/td&gt;&lt;/tr&gt;";
33              }
34          ?&gt;
35      &lt;/table&gt;
36      &lt;input type="hidden" name="action" value="savePerms" /&gt;
37      &lt;input type="hidden" name="userID" value="<?= $_GET['userID']
38      &lt;input type="submit" name="Submit" value="Submit" /&gt;
39      &lt;input type="button" name="Cancel" onclick="window.location='?act
40  &lt;/form&gt;
41  &lt;? } ?&gt;
</pre

```

Like the roles form, we start by adding a form and table, this time with 2 columns. Then we create the ACL object, pull the permissions array (line 8), and get an array of all the permissions (line 9). In the **foreach()** loop we print out a new row and the name of the permission. Then we start a select element. The select input will have 3

options: Allow, Deny and Inherit. We look at the value of `$rPerms[$v['Key']]['value']` to see which option should be selected. Allow or Deny will not be selected if the permission value is inherited thanks to `$rPerms[$v['Key']]['inherited'] != true`. If the permission is inherited, the Inherited option will be selected.

Line 23-32 enhance the inherit option. If the permission is inherited, it makes it selected. Then it determines the value of the inherited permission and sets the variable `$iVal` so we can use the text value in the option label on line 33. After ending the select input and the table, we add in the hidden inputs to set up the save options, and add submit and cancel buttons.

Once this code is run, we will end up with a row for each available permission, and a drop down indicating whether or not the user has it.

My Site

Doing Stuff Since 1900

[Main Screen](#) | [Admin Home](#)

Manage User Permissions: (Admin Steve)

Access Admin System	<input type="text" value="Inherit (Allow)"/>
Access Premium Content	<input type="text" value="Inherit (Allow)"/>
Access Site	<input type="text" value="Inherit (Allow)"/>
Install Modules	<input type="text" value="Inherit (Allow)"/>
Limited Admin	<input type="text" value="Inherit (Allow)"/>
Post Comments	<input type="text" value="Inherit (Allow)"/>
Publish Articles	<input type="text" value="Inherit (Allow)"/>
Publish Events	<input type="text" value="Inherit (Allow)"/>

Submit

Cancel

Saving the Data

Add this code to `/admin/users.php` right above the doc type tag:

```

01 <? if (isset($_POST['action']))
02 {
03     switch($_POST['action'])
04     {
05         case 'saveRoles':
06             $redirect = "?action=user&userID=" . $_POST['userID'];
07             foreach ($_POST as $k => $v)
08             {
09                 if (substr($k,0,5) == "role_")
10                 {
11                     $roleID = str_replace("role_", "", $k);
12                     if ($v == '0' || $v == 'x') {
13                         $strSQL = sprintf("DELETE FROM `user_role`");
14                     } else {
15                         $strSQL = sprintf("REPLACE INTO `user_role`");
16                     }
17                     mysql_query($strSQL);
18                 }
19             }
20
21             break;
22         case 'savePerms':
23             $redirect = "?action=user&userID=" . $_POST['userID'];
24             foreach ($_POST as $k => $v)
25             {
26                 if (substr($k,0,5) == "perm_")
27                 {
28                     $permID = str_replace("perm_", "", $k);
29                     if ($v == 'x')
30                     {
31                         $strSQL = sprintf("DELETE FROM `user_perm`");
32                     } else {
33                         $strSQL = sprintf("REPLACE INTO `user_perm`");
34                     }
35                     mysql_query($strSQL);
36                 }
37             }
38             break;
39         }
40         header("location: users.php" . $redirect);
41     }
42 }
?>

```

This code first checks to see if something has been submitted by looking at **\$_POST['action']**. This is the value that was in one of the hidden form elements in the two forms we made.

If we just submitted the roles form, the following happens:

1. We build a **\$redirect** querystring which is where we will be sent after the form processes.
2. We loop through all of the **\$_POST** variables.
3. Using **substr()** we find out if the first 5 digits of the variable name are "role_".

This way we only get the permission inputs in the following steps.

4. If the value for the current input is equal to '0' or 'x' (i.e. we don't want the user to have this role), we perform the delete query. If we delete the role from the user_roles table, the user is no longer assigned to the role.
5. If the value is not '0' or 'x' (line 14), we perform the replace query.
6. For either query, we are using **sprintf()** for security (**sprintf()** forces variable typing and helps protect against SQL injection attacks [more info](#)).
7. We execute the SQL using **mysql_query()**.

Note on the replace query: The replace syntax is a special MySQL syntax that allows a seamless update or insert. By using the replace, it can save us from writing lots of PHP code. When we created the user_roles table, we created a unique index on the userID and roleID fields. When we execute the 'replace into' statement, it first looks in the table to see if inserting a new row would create a duplicate (i.e. a row with the same index values already exists). If there is a row that matches the indexes, it updates that row. If there isn't, it inserts a new row. For more info, see the [MySQL developer site](#).

If we just submitted the permissions form, the process is the same, except we are looking for a different prefix on the input names, and using a different database table. Once any operations are done, we use **header("location:...")** to redirect back to the page we were on, and we append the **\$redir** querystring variable we made.

Step 5: Roles Admin

Now that we have finished the forms to manage our users, we need to manage our roles. The roles will be more simple, there are only two actions: view a list of roles, or edit a role. Create /admin/roles.php with the following code:

```
01 <?php
02 include("../assets/php/database.php");
03 include("../assets/php/class.acl.php");
04 $myACL = new ACL();
05 if ($myACL->hasPermission('access_admin') != true)<br />{<br />
06     ?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
07     <html xmlns="http://www.w3.org/1999/xhtml">
08     <head>
09     <meta http-equiv="Content-Type" content="text/html; charset=utf-8
10     <title>ACL Test</title>
11     <link href="../assets/css/styles.css" rel="stylesheet" type="text
12     </head>
13     <body>
```

```

14 <div id="header"></div>
15 <div id="adminButton"><a href="..">Main Screen</a> | <a href="in
16 <div id="page">
17     <!-- PAGE CONTENT -->
18 </div>
19 </body>
20 </html>

```

List Roles

Like the users page, we start off with the includes, creating the ACL object, and the page format. Our default action (page loaded with no querystring) is to list the available roles, so insert this code in place of `<!-- PAGE CONTENT -->`:

```

01 <? if ($_GET['action'] == '') { ?>
02     <h2>Select a Role to Manage:</h2>
03     <?
04     $roles = $myACL->getAllRoles('full');
05     foreach ($roles as $k => $v)
06     {
07         echo "<a href=\"?action=role&roleID=" . $v['ID'] . "\">"
08     }
09     if (count($roles) < 1)
10     {
11         echo "No roles yet.<br />";
12     } ?>
13     <input type="button" name="New" value="New Role" onclick="win
14 <? } ?>

```

First we check if the querystring var was empty. Then we store a list of all the available roles in `$roles` by using `getAllRoles()`. On each iteration of the `foreach()` loop, we make a link that will bring us to the form to edit an individual role. If there are no roles in the `$roles` array, we display a friendly message. Finally, we add in a button that will allow us to add a new role.

Edit Role

Add this code in `/admin/roles.php` under the previous block:

```

01 <? if ($_GET['action'] == 'role') {
02     if ($_GET['roleID'] == '') {
03         ?>
04         <h2>New Role:</h2>
05         <? } else { ?>
06         <h2>Manage Role: (= $myACL-&gt;getRoleNameFromID($_GET['roleID']
07         &lt;form action="roles.php" method="post"&gt;
08             &lt;label for="roleName"&gt;Name:&lt;/label&gt;&lt;input type="text" nam
09             &lt;table border="0" cellpadding="5" cellspacing="0"&gt;
10
</pre

```

```

11      <tr><th></th><th>Allow</th><th>Deny</th><th>Ignore</th></tr></th></tr>
12      <?
13      $rPerms = $myACL->getRolePerms($_GET['roleID']);
14      $aPerms = $myACL->getAllPerms('full');
15      foreach ($aPerms as $k => $v)
16      {
17          echo "<tr><td><label>" . $v['Name'] . "</label></td>"
18          echo "<td><input type=\"radio\" name=\"perm_\" . $v['I
19          if ($rPerms[$v['Key']]['value'] === true && $_GET['ro
20          echo " /></td>";
21          echo "<td><input type=\"radio\" name=\"perm_\" . $v['I
22          if ($rPerms[$v['Key']]['value'] != true && $_GET['rol
23          echo " /></td>";
24          echo "<td><input type=\"radio\" name=\"perm_\" . $v['I
25          if ($_GET['roleID'] == '' || !array_key_exists($v['Ke
26          echo " /></td>";
27          echo "</tr>";
28      }
29      ?>
30      </table>
31      <input type="hidden" name="action" value="saveRole" />
32      <input type="hidden" name="roleID" value="<?= $_GET['roleID']
33      <input type="submit" name="Submit" value="Submit" />
34  </form>
35  <form action="roles.php" method="post">
36      <input type="hidden" name="action" value="delRole" />
37      <input type="hidden" name="roleID" value="<?= $_GET['roleID']
38      <input type="submit" name="Delete" value="Delete" />
39  </form>
40  <form action="roles.php" method="post">
41      <input type="submit" name="Cancel" value="Cancel" />
42  </form>
43  <? } ?>

```

After checking to make sure the querystring variable is there, we see if a roleID was passed in the querystring. If one was, we assume that we are editing a role, if not we are creating one (we display a header as appropriate). Then we create a form. Inside the form, we need a text input for the name of our role, and a table to hold the permissions. The table has columns for the permission name, allow, deny, and ignore. Like we did while editing user permissions, we must loop through the array of all permissions (line 15, `$myACL->getAllPerms('full')`)

In each row, we print the permission name, and 3 radio buttons. The radios use the same nomenclature as the user form ("perm_[permID]"). 'Allow' or 'Deny' are selected depending on the value of the permission stored (thanks to lines 19 and 22). If you select 'ignore', no value is stored for that role/permission combo. Notice that the first two `if()` block have `&& $_GET['roleID'] != ''` in them. This ensures that if no user ID is passed (that we are creating a new role), ignore is selected by default. Then we add the hidden inputs to set the save options, and close the form. We also add another form with hidden inputs to delete the role, and another form with a cancel

button that will return us to the roles page.

If everything went according to plan, we should get the following when we try to edit the permissions for a role:

Saving the Data

Insert this code in `/admin/roles.php` right before the doc type tag:

```

01  <? if (isset($_POST['action']))
02  {
03      switch($_POST['action'])
04      {
05          case 'saveRole':
06              $strSQL = sprintf("REPLACE INTO `roles` SET `ID` = %u
07              mysql_query($strSQL);
08              if (mysql_affected_rows() > 1)
09              {
10                  $roleID = $_POST['roleID'];
11              } else {
12                  $roleID = mysql_insert_id();
13              }
14              foreach ($_POST as $k => $v)
15              {
16                  if (substr($k,0,5) == "perm_")
17

```



```

18         {
19             $permID = str_replace("perm_", "", $k);
20             if ($v == 'X')
21             {
22                 $strSQL = sprintf("DELETE FROM `role_perm`
23                 mysql_query($strSQL);
24                 continue;
25             }
26             $strSQL = sprintf("REPLACE INTO `role_perms`
27             mysql_query($strSQL);
28         }
29     }
30     header("location: roles.php");
31     break;
32     case 'delRole':
33         $strSQL = sprintf("DELETE FROM `roles` WHERE `ID` = %
34         mysql_query($strSQL);
35         $strSQL = sprintf("DELETE FROM `user_roles` WHERE `ro
36         mysql_query($strSQL);
37         $strSQL = sprintf("DELETE FROM `role_perms` WHERE `ro
38         mysql_query($strSQL);
39         header("location: roles.php");
40     break;
41 }
42 }
43
?>

```

Like on the users page, we check to see if something was submitted via **\$_POST**, and what the value of **\$_POST['action']** was. If we were saving a role, we do the following:

1. Perform a replace query on the roles table. This will update/insert the role name. Lines 8-13 perform an important function for saving roles. If we are performing an update, we already have an ID for the role. However if we are inserting one, we don't know the role ID. When we perform the replace query, the number of rows affected are returned. If the number of rows affected was greater than 1, a row was updated, so we should use the role id from the form. If the rows affected was not greater than 1, the row was inserted, so we use **mysql_insert_id()** to get the ID for the last inserted row.
2. Then we loop through the **\$_POST** variables and line 16 ensures that we only process rows where the input name starts with "perm_".
3. Line 18 gets the **floatval()** of the permission so we end up with just the integer ID of the perm (so we know which permission we are dealing with).
4. **if (\$v == 'x') {...}** will run if we selected 'Ignore' for a permission on the form. It will attempt to delete the row from the table where the row ID and permission ID are right. If this happens, we use **continue**; to go to the next variable.
5. If we have gotten to this point, we assume that we want to add or update a

permission for this role. So, we use the 'replace into' syntax that we used in the user form. It's important that we have the roleID and permID in there so the database can check for an existing row.

6. Finally we execute the SQL and redirect to the roles page.

If we have submitted the delete form, we delete the role from the roles table. Then we also delete any records from the user_roles and role_perms tables that match the role ID so that we don't end up with users and permissions assigned to roles that don't exist. Then we redirect to the roles page.

Step 6: Permissions Admin

Like the roles admin, the permissions admin will have two functions: list the available permissions, and editing permissions. Start with this code in /admin/perms.php:

```
01 <?php
02 include("../assets/php/database.php");
03 include("../assets/php/class.acl.php");
04 $myACL = new ACL();
05 if ($myACL->hasPermission('access_admin') != true)<br />{<br />
06 ?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
07 <html xmlns="http://www.w3.org/1999/xhtml">
08 <head>
09 <meta http-equiv="Content-Type" content="text/html; charset=utf-8
10 <title>ACL Test</title>
11 <link href="../assets/css/styles.css" rel="stylesheet" type="text
12 </head>
13 <body>
14 <div id="header"></div>
15 <div id="adminButton"><a href="..">Main Screen</a> | <a href="in
16 <div id="page">
17     <!-- PAGE CONTENT -->
18 </div>
19 </body>
20 </html>
```

List Permissions

Place this code in the page div (in place of <!-- PAGE CONTENT -->):

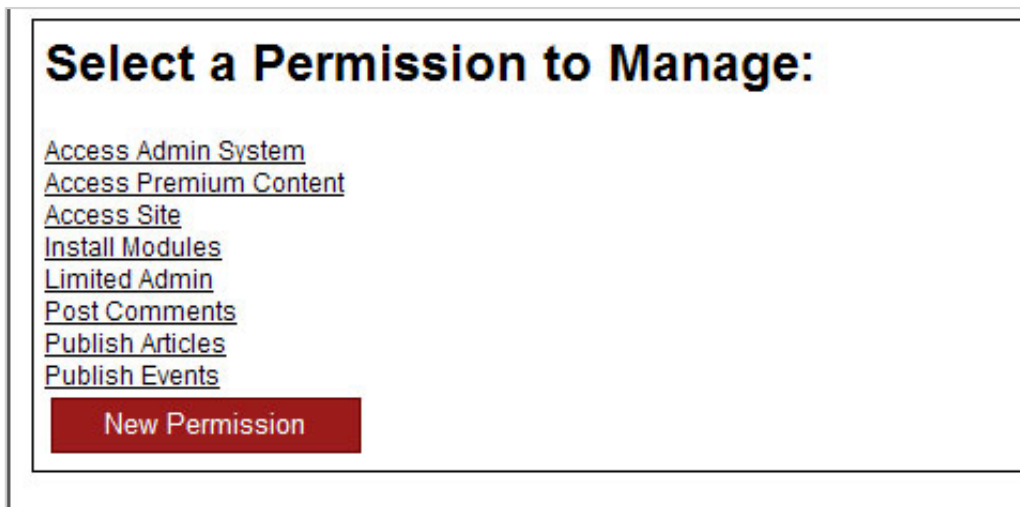
```
01 <? if ($_GET['action'] == '') { ?>
02     <h2>Select a Permission to Manage:</h2>
03     <?
04     $roles = $myACL->getAllPerms('full');
05     foreach ($roles as $k => $v)
06     {
```

```

07         echo "<a href=\"?action=perm&permID=" . $v['ID'] . "\">"
08     }
09     if (count($roles) < 1)
10     {
11         echo "No permissions yet.<br />";
12     } ?>
13     <input type="button" name="New" value="New Permission" onclick
14     <? } ?>

```

We will first use **getAllPerms()** to get an array of all the permissions. Then we will loop through it to build our list. Each iteration through the **foreach()** loop will generate a link that will direct us to the page to edit the given permission. If no permissions are present, we display a message saying so, and we end the form with a 'New Permission' button. And the result:



Edit Permission

To edit/add an individual permission, we need to add this code immediately after the previous block:

```

01 <? if ($_GET['action'] == 'perm') {
02     if ($_GET['permID'] == '') {
03         ?>
04         <h2>New Permission:</h2>
05         <? } else { ?>
06         <h2>Manage Permission: (<?= $myACL->getPermNameFromID($_GET['
07         <form action="perms.php" method="post">
08             <label for="permName">Name:</label><input type="text" nam
09             <label for="permKey">Key:</label><input type="text" name=
10             <input type="hidden" name="action" value="savePerm" />
11             <input type="hidden" name="permID" value="<?= $_GET['permID']
12             <input type="submit" name="Submit" value="Submit" />
13         </form>
14         <form action="perms.php" method="post">
15             <input type="hidden" name="action" value="delPerm" />
16             <input type="hidden" name="permID" value="<?= $_GET['permID'

```

```
17     <input type="submit" name="Delete" value="Delete" />
18 </form>
19 <form action="perms.php" method="post">
20     <input type="submit" name="Cancel" value="Cancel" />
21 </form>
22 <? } ?>
```

Like we did in the roles form, we check to see if a permission ID is provided in the querystring and display either an addition or update header based on that. We open a form tag, and add two text inputs: one for the permission name, the other for the permission key. The name is what will appear on forms, while the key is what will be used in scripts. The key should be pretty much the same as the name, except for it should not have spaces or symbols, and should be lower case. For both text fields, we provide default values if we are updating.

At the end of the form, we add the hidden inputs, and the submit button. Then we have the delete and cancel forms.

Save the Data

Finally, we need to save the permission form, so add this code to the top of /admin/perms.php right above the doc type.

```
01  if (isset($_POST['action']))
02  {
03      switch($_POST['action'])
04      {
05          case 'savePerm':
06              $strSQL = sprintf("REPLACE INTO `permissions` SET `ID`
07              mysql_query($strSQL);
08              break;
09          case 'delPerm':
10              $strSQL = sprintf("DELETE FROM `permissions` WHERE `I
11              mysql_query($strSQL);
12              break;
13      }
14      header("location: perms.php");
15  }
```

Like all the other submission scripts, we need to figure out what action was submitted. If we are saving a permission, we perform a replace into operation. This will either update or insert as appropriate. If we submitted the delete form, we perform the delete query. In either case, we will be redirected to perms.php.

Step 7: Admin hub

We need a jumping off point for our ACL admin. We'll just create something simple with links to the 3 pages. Here is a preview and the code for it:



```

01  <?php
02  include("../assets/php/database.php");
03  include("../assets/php/class.acl.php");
04  $myACL = new ACL();
05  if ($myACL->hasPermission('access_admin') != true)<br />{<br />
06  ?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
07  <html xmlns="http://www.w3.org/1999/xhtml">
08  <head>
09  <meta http-equiv="Content-Type" content="text/html; charset=utf-8
10  <title>ACL Test</title>
11  <link href="../assets/css/styles.css" rel="stylesheet" type="text
12  </head>
13  <body>
14  <div id="header"></div>
15  <div id="adminButton"><a href="..">Main Screen</a></div>
16  <div id="page">
17      <h2>Select an Admin Function:</h2>
18      <a href="users.php">Manage Users</a><br />
19      <a href="roles.php">Manage Roles</a><br />
20      <a href="perms.php">Manage Permissions</a><br />
21  </div>
22  </body>
23  </html>

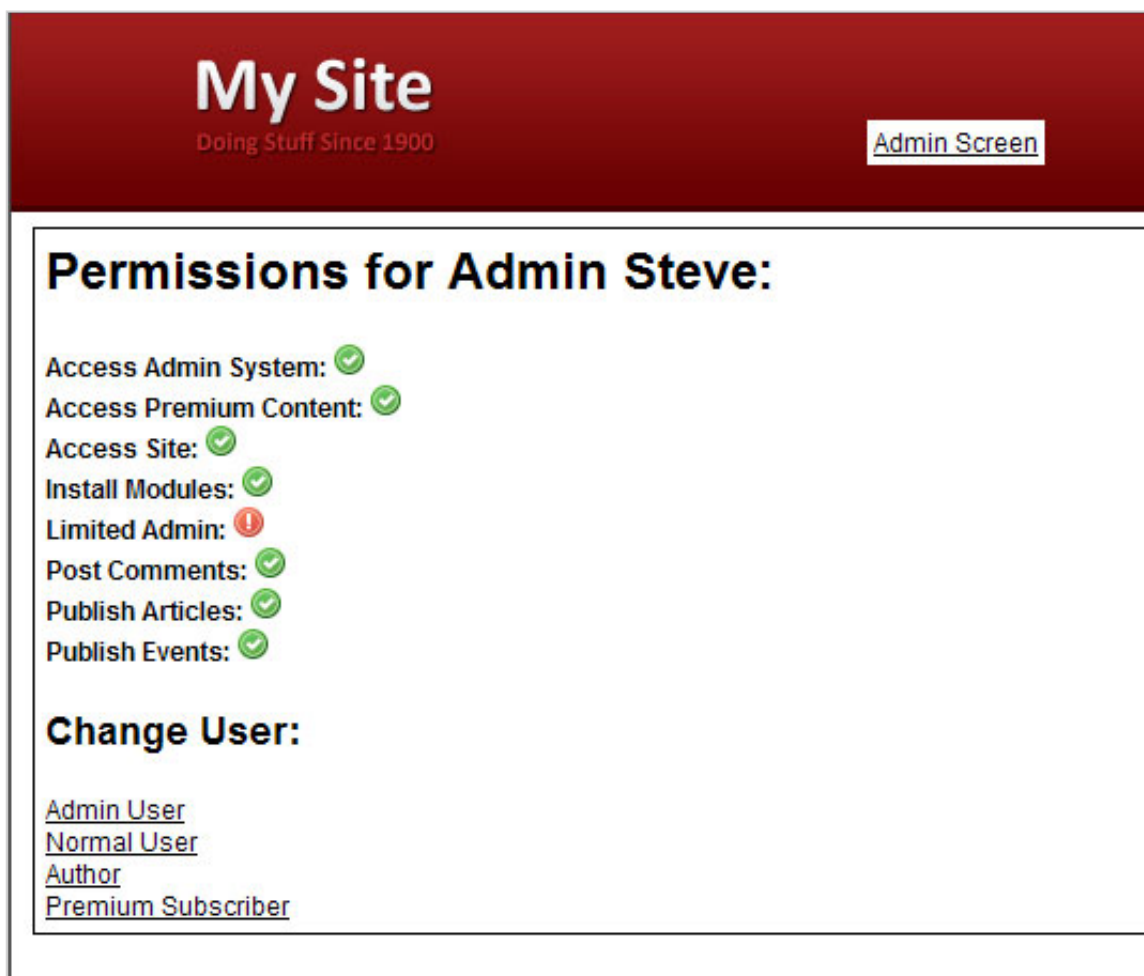
```

Pretty self-explanatory, we have 3 links to manage the 3 different aspects of your ACL.



Advertisement

Step 8: Implementing the ACL on Your Site



Implementing your new ACL system on your site is fairly easy. Each page which you want to secure should have the database and ACL file included at the top. After that, you should create a new instance of the ACL object.

As an example, say you had set up a permission with the key 'access_admin' and

wanted to use it to control access to the admin interface. At the top of your page you could use this script to check it:

```

1  <?php
2  include("assets/php/database.php");
3  include("assets/php/class.acl.php");
4  $myACL = new ACL();
5  if ($myACL->hasPermission('access_admin') != true)
6  {
7      header("location: insufficientPermission.php");
8  }
9  ?>

```

As you can see we create an ACL object. Since we are not passing in a user ID as an argument, the system will read the session variable `$_SESSION['userID']`. Then we use `$myACL->hasPermission('access_admin')` to check to see if the user has that permission. If they do not, they are redirected to `insufficientPermission.php`. This way they can't get in to secure areas that they don't have permissions for.

In the provided source files, I have provided an index file that provides a simple test of the ACL based on the example code above. The sample index displays a list of all the permissions, and icons representing whether or not the current user can access each. There is also a list of the users that allows you to change the user that the ACL is displayed for. Here is the code for the sample index:

```

01  <?php
02  include("assets/php/database.php");
03  include("assets/php/class.acl.php");
04
05  $userID = $_GET['userID'];
06  $_SESSION['userID'] = 1;
07  $myACL = new ACL();
08  ?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
09  <html xmlns="http://www.w3.org/1999/xhtml">
10  <head>
11  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
12  <title>ACL Test</title>
13  <link href="assets/css/styles.css" rel="stylesheet" type="text/css" />
14  </head>
15  <body>
16  <div id="header"></div>
17  <div id="adminButton"><a href="admin/">Admin Screen</a></div>
18  <div id="page">
19      <h2>Permissions for <?= $myACL->getUsername($userID); ?>:</h2>
20      <?
21          $userACL = new ACL($userID);
22          $aPerms = $userACL->getAllPerms('full');
23          foreach ($aPerms as $k => $v)
24          {
25              echo "<strong> " . $v['Name'] . ": </strong>";

```

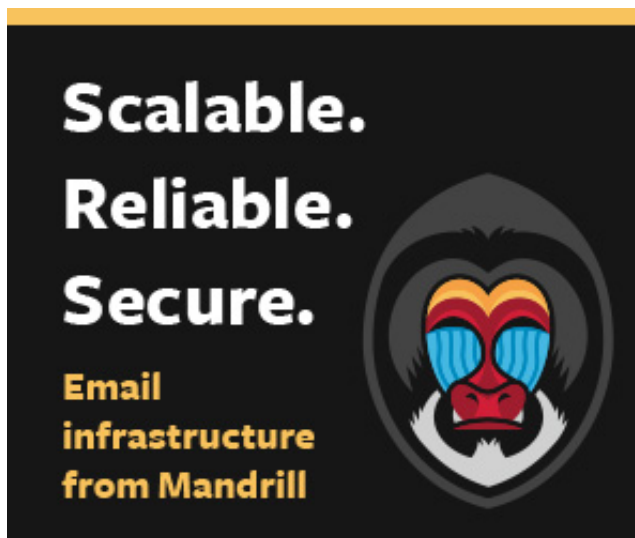


```
26         echo "<img src=\"assets/img/";
27         if ($userACL->hasPermission($v['Key']) === true)
28         {
29             echo "allow.png";
30             $pVal = "Allow";
31         } else {
32             echo "deny.png";
33             $pVal = "Deny";
34         }
35         echo "\" width=\"16\" height=\"16\" alt=\"$pVal\" /><
36     }
37 ?>
38 <h3>Change User:</h3>
39 <?
40     $strSQL = "SELECT * FROM `users` ORDER BY `Username` ASC"
41     $data = mysql_query($strSQL);
42     while ($row = mysql_fetch_assoc($data))
43     {
44         echo "<a href=\"?userID=" . $row['ID'] . "\">" . $row
45     }
46 ?>
47 </div>
48 </body>
49 </html>
```

Final Thoughts

When combined with a good user management platform, an ACL system is a great way to secure your web site. By following these steps, you should be able to create your own flexible security system. The admin system created here is a basic example of what you can create if you don't already have an admin system set up. It demonstrates all of the principles you need to effectively manage your ACL. On the other hand, if you already have created your own user management system, it should be fairly easy to take these techniques and implement them into your own project.

Subscribe to the [NETTUTS RSS Feed](#) for more daily web development tuts and articles.



Advertisement

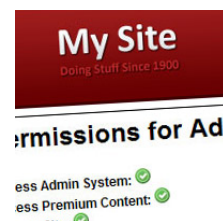
*Difficulty:***Intermediate***Length:***Medium***Categories:*

PHP



Web Development

Translations Available:

Tuts+ tutorials are translated by our community members. If you'd like to translate this post into another language, [let us know!](#)



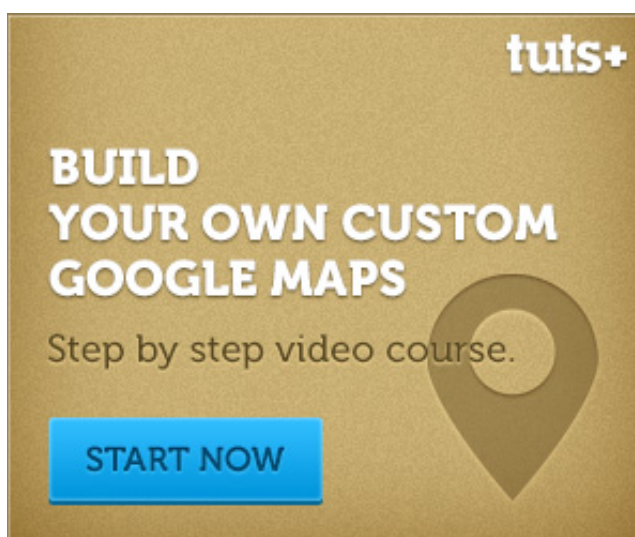
Permissions for Ad

Less Admin System: 
Less Premium Content: 

About Andrew Steenbuck



N/A



Advertisement

Suggested Tuts+ Course



[Getting Started With Symfony 2](#)

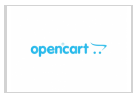
\$15

Related Tutorials



[Creating a Custom WordPress Registration Form Plugin](#)

[Code](#)



[From Beginner to Advanced in OpenCart: Module Development](#)

[Code](#)



[The Privacy and Optimization of a WordPress Dashboard Widget](#)

[Code](#)

Jobs

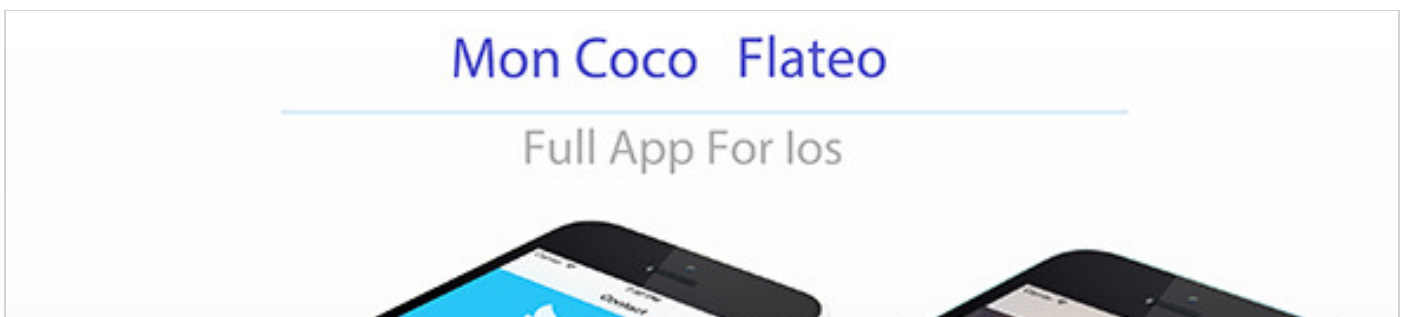


[WordPress Designer/Developer
at Killer Rezzzy in New York, NY, USA](#)



[PHP/WordPress Developer
in West Palm Beach, FL, USA](#)

Envato Market Item





Join InMotion Hosting
for \$48 & get a year of
learning on Tuts+ **FREE**
(worth \$180).

Start today.

inmotion.
hosting

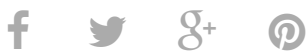
tuts+
by envato

Advertisement

Teaching skills to millions worldwide.

18,414 Tutorials 420 Video Courses

Follow Us



Help and Support

[FAQ](#)

[Terms of Use](#)

[Contact Support](#)

[About Tuts+](#)

[Advertise](#)

[Write for Us](#)

Email Newsletters

Subscribe to receive inspiration, ideas,
and news in your inbox.

Email Address

Subscribe

[Privacy Policy](#)



Custom digital services like logo design, WordPress installation, video production and more.

[Check out Envato Studio](#)



Add more features to your website such as user profiles, payment gateways, image galleries and more.

[Browse WordPress Plugins](#)

© 2014 Envato Pty Ltd. Trademarks and brands are the property of their respective owners.

