

Robust Permissions for Web Applications

Brandon Savage ZendCon 2008

Good evening.

I hope that everyone has enjoyed today.

Today I'm discussing Robust Permissions for Web Applications.

This is a talk on a library that I developed for a client, and it has somewhat grown into a robust library that I use for my own work. I hope that you will too.

So why this library?

Web applications are growing larger.



Every day, web applications are growing larger.

What started as small guest books have morphed into large-scale websites like Facebook and Craigslist.

Zend's CEO Harold Goldberg reminded us that PHP runs almost a third of the web.

With growing websites, comes a growing need for an easy way to manage permissions.

As developers, we need ways to control access.

Truth be told, this isn't an easy challenge. People are placing on us, as developers, new demands for new methods. And as developers, it is our job to ensure that only the right people have access.

This isn't easy, because as the complexity of an application increases, so does the complexity of the permissions scheme.

Increasing complexity means increasingly complex permissions.

Just take a look at Facebook. Facebook's privacy settings are exceptionally complex, with sliders that indicate how serious your privacy settings are set to be.

This scheme of sliders was created to combat an increasingly complicated scheme of assigning permissions and rights. But as Facebook gives their users more granular control, the complexity creep has increased.


We want a simple method for managing complex permissions.

Simply, we really want to find a way to simply manage permissions, regardless of their complexity.

We want a way to assign users into groups, to assign group-level permissions, and then control individual users.

We want to cache these permissions in order to reduce the load on the server.

And we certainly want to make sure that the implementation and management of the permissions interface is simple, seamless, and easy.



ApplicationACL provides a simple method.

ApplicationACL provides the solution to these wants and needs.

It's easy to use. It's simple to install. And most importantly, it's scalable for large applications, meaning you don't have to worry about whether or not it will crash under large load.

Let's take a closer look.

ApplicationACL is easy to use.



First, I want to show you how easy it is to use.

ApplicationACL is easy to use because it's designed for simple, quick administration. Taking after some of the best (and worst) models, we've come up with a simple interface for management.

Management interface is a piece of cake.



The truth is, the management interface for ApplicationACL is a piece of cake.

A lot of time went into designing this from the user standpoint, and I think that your implementation will show you that this is true.

	<input type="radio"/> + <input type="radio"/> r <input type="radio"/> -	<input type="radio"/> + <input type="radio"/> w <input type="radio"/> -	<input type="radio"/> + <input type="radio"/> c <input type="radio"/> -	<input type="radio"/> + <input type="radio"/> d <input type="radio"/> -
test_application	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	<input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>

Designed with radio buttons for easy configuration

We wanted to give you a simple table-based interface for granting or denying permissions.

So, as you'll note from the image, each application is listed, and then there are a series of radio buttons and graphics that show the permissions level for this application and this user.

It's obvious from the screen shot that this user has not been assigned read or write permissions, but has been denied create and delete permissions.

Management comes with intuitive table interface.

I also spent a while working through just how I was going to add and remove applications, and there is a sample included with the download that shows this.

Let's take a look at how this all works in a fully developed environment.

Let's see it in action.

DEMO

There are only two methods for answering a permissions query.

Permissions Returns

```
boolean false
```

```
array
```

```
'r' => int 0  
'w' => int 0  
'c' => string '-1' (length=2)  
'd' => string '-1' (length=2)
```

ApplicationACL is designed primarily as a black box. That means that you send information into it, and you get information back out. As such, the methods are somewhat limited for querying data.

Let's examine the methods that we have.

#1. You can return a boolean result.

Boolean Results

```
$obj->return_user_permission('1','test_application','r');
```

```
boolean false
```

First, you can get a boolean result back.

This involves giving it the user, the object, and the permissions level that you're investigating.

It will then query the database (or the cache) and return true or false based on whether or not this user has a recorded permissions level.

It makes no distinction as to whether the user is denied or simply has not been granted access; that's done later.

This method might not always work for us, and might be inefficient. Let's take a look at the second method.

Array Results

```
$obj->return_user_array('1','test_application');
```

```
array
  'r' => int 0
  'w' => int 0
  'c' => string '-1' (length=2)
  'd' => string '-1' (length=2)
```

#2. You can return an array.

Our second option is to return an array of permissions.

In this scenario, we only ask for the user and the object id. We're going to get an entire array of this user's permissions for this particular item or application.

As you can see, we've got an array of the permissions for all the permissions, rather than only one.



**It can interface with any
database object.**

One of the best features of ApplicationACL is that it's designed and pre-built to interface with your existing database object or database abstraction layer.

```
// MYSQL CREDENTIALS  
define('DBO', 'DBObject'); // What is the class name of the database  
define('DBO_QUERY_METHOD', 'query'); // What method does the DBO use  
define('DATABASE', 'permissions'); // What database are these files
```

Simply designate the name of the object in the settings...

First, you define the database object constant...

```
// MYSQL CREDENTIALS
define('DBO', 'DBObject'), // What is the class name of the database
define('DBO_QUERY_METHOD', 'query'); // What method does the DBO use
define('DATABASE', 'permissions'); // What database are these files
```

...and the name of the method
accepting custom queries.

And then define the database query method that executes a
straight SQL query, and returns a result.

ApplicationACL was
designed to be
portable.

The upshot is that ApplicationACL was designed to be portable.

By allowing you to use any database object and defining the query used, you can take the application out of the application its in, and move it over to the application that needs it.

ApplicationACL is simple to install.



The second thing I wanted to highlight is that ApplicationACL is simple to install.

Very simple. We'll see how simple very soon.

The beauty of ApplicationACL's configuration and easy setup is in its settings file.

```
// MYSQL CREDENTIALS
define('DBO','DBObject'); // What is the class name of the database object?
define('DBO_QUERY_METHOD','query'); // What method does the DBO use to accept a straight SQL qu
define('DATABASE','permissions'); // What database are these files stored in?

// TABLE INFORMATION
define('USER_TABLE','users'); // What table are users stored in?
define('USER_PERM_TABLE','user_permissions'); // What table are user permissions stored in?
define('GROUP_PERM_TABLE','group_permissions'); // What table are group permissions stored in?
define('USER_GROUP_TABLE','user_to_group'); // What table do we use to join users into groups?
```

The settings file is clear,
concise and well-documented.

The first thing to note is the composition of the settings file.

Note it's cleanliness and use of defined constants to set the settings.

The file is designed so that it's easy to use and configure with little or no knowledge of the application as a whole.

```

1 <?php
2
3 /*****
4 * SETTINGS
5 * This defines the settings for the Permissions library.
6 *
7 * These settings are used in the classes to ensure that y
8 * honored.
9 *
10 *****/
11
12 // MYSQL CREDENTIALS
13 define('DBO', 'DBObject'); // What is the class name of th
14 define('DBO_QUERY_METHOD', 'query'); // What method does t
15 define('DATABASE', 'permissions'); // What database are th
16
17 // TABLE INFORMATION
18 define('USER_TABLE', 'users'); // What table are users sto
19 define('USER_PERM_TABLE', 'user_permissions'); // What tab
20 define('GROUP_PERM_TABLE', 'group_permissions'); // What t
21 define('USER_GROUP_TABLE', 'user_to_group'); // What table
22
23 // TABLE FIELD INFORMATION
24 define('USER_TABLE_ID', 'id'); // What is the ID field cal

```

All settings
are
configured
as constants.

The settings file is composed of constants.

These constants define the information for the program. Since the constants work inside the classes, it creates a simple and easy way to set up the application.

```
protected $user_table = USER_TABLE;  
protected $user_perm_table = USER_PERM_TABLE;  
protected $group_perm_table = GROUP_PERM_TABLE;  
protected $user_group_table = USER_GROUP_TABLE;  
  
protected $user_table_id = USER_TABLE_ID;  
protected $user_group_table_user = USER_GROUP_TABLE_USER;  
protected $user_group_table_group = USER_GROUP_TABLE_GROUP;  
protected $user_id_perms = USER_ID_PERMS;  
protected $group_id_perms = GROUP_ID_PERMS;  
protected $object_id_perms = OBJECT_ID_PERMS;  
  
protected $user_permissions = USER_PERMISSIONS;  
protected $group_permissions = GROUP_PERMISSIONS;
```

Each constant is then converted into a property.

In order to aid in proper OOP design, each constant is converted into a property.

This is so that they can be accessed with the `$this` keyword. You'll thank me if you ever dig into the code to edit it.

Set the settings once, and forget about them.

The great thing about the settings file is that you can 'set it and forget it.'

Since the properties are defined for you, you don't have to define them every time you instantiate the object.

No database
redesign is
required.

































One of the most exciting things about this is that it doesn't require you to redesign your database.

ApplicationACL integrates with the tables you already have.

localhost ▶ permissions

Structure SQL Search Query Export Import Operations Privileges Drop

	Table	Action	Records	Type	Collation	Size	Overhead
<input type="checkbox"/>	applications	     	1	MyISAM	latin1_swedish_ci	2.0 KiB	-
<input type="checkbox"/>	group_permissions	     	0	MyISAM	latin1_swedish_ci	1.0 KiB	-
<input type="checkbox"/>	users	     	0	MyISAM	latin1_swedish_ci	1.0 KiB	-
<input type="checkbox"/>	user_permissions	     	1	MyISAM	latin1_swedish_ci	4.0 KiB	-
<input type="checkbox"/>	user_to_group	     	0	MyISAM	latin1_swedish_ci	1.0 KiB	-
5 table(s)		Sum	2	MyISAM	latin1_swedish_ci	9.1 KiB	0 B

Check All / Uncheck All

With selected:

The settings file lets you identify your current tables.

The settings file contains constants that let you define your existing user and group tables – or the names of tables you'd like it to create.

The purpose for this is to make sure that you can integrate this into an existing application. Obviously, you've already got a users table, and I wouldn't want you to have to redesign the database.



Missing
database
tables can be
automatically
generated.

Let's say for a moment that you decide you want to install this application and you don't have a groups table. What do you do?

The script is designed that if you ask it to do so, it will automatically create the tables for you.

This feature is particularly useful, and let's take a look at it in action.

With
experience,
you can
install it in 5
minutes or
less.



With some experience, and an intimate knowledge of your application, you can install this in five minutes or less.

The Five Minute Install.

DEMO

ApplicationACL is scalable.

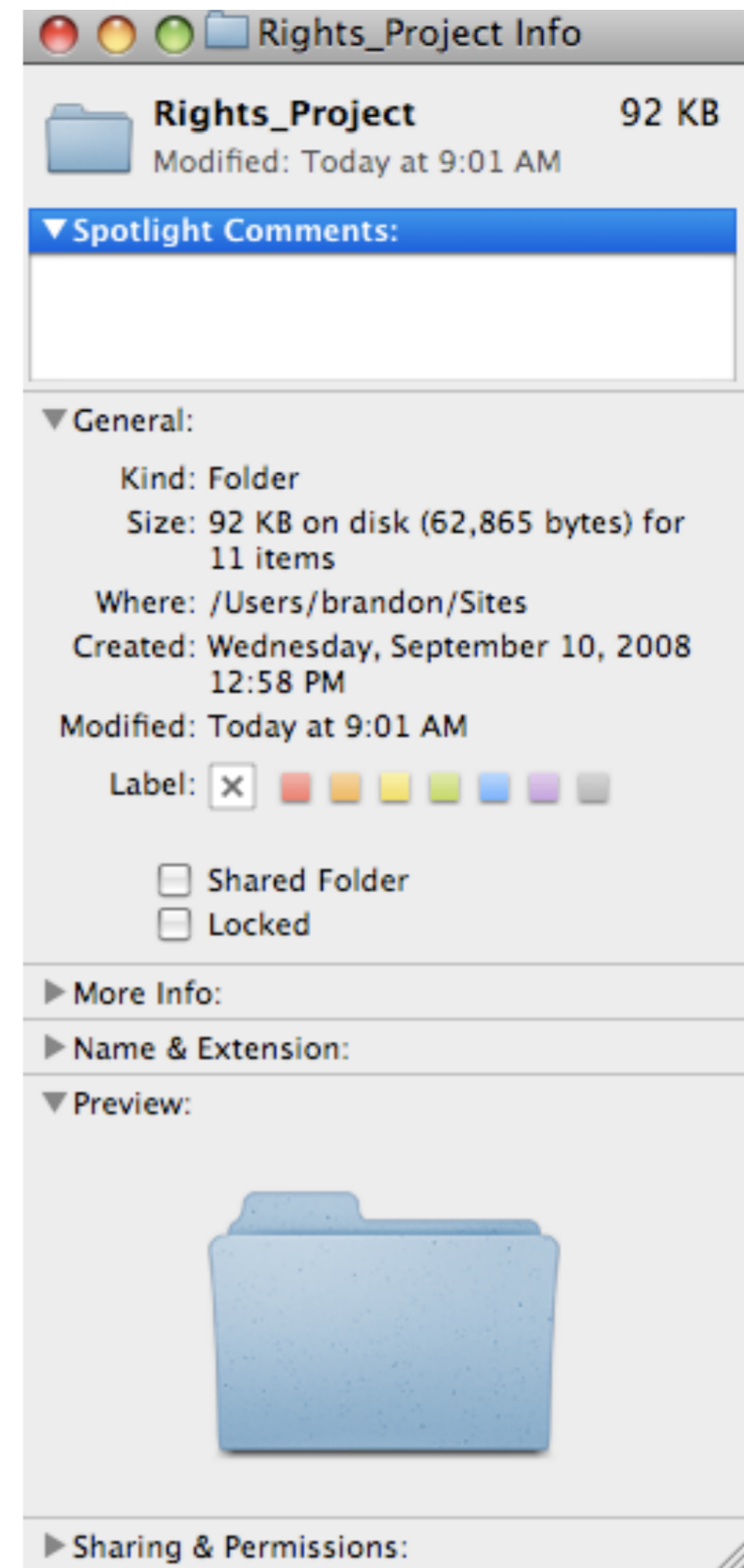


First, I want to say that no, this is **NOT** a comment about Ruby on Rails...Rails, sorry, Rails.

The third point I want to make today about ApplicationACL is that it's scalable. Regardless of your application, it can be made to scale with the use of caching.

Additionally, it's small code base leaves a small footprint, which is a benefit over larger frameworks that might provide only a few useful features depending on the application.

With a codebase of less than 100KB, the application is small.



First, the easy stuff.

The codebase is less than 100kb. With the images, it's a little over 104 kb.

This small footprint allows it to be integrated with little added overhead.


```
interface CacheRightsInterface

# Not defined, but recommend a constructor function.

# Have a way to cache data.
public function put_in_cache($key,$object_id,$rights);

# Have a way to retrieve cached data.
public function get_from_cache($key,$object_id,$rights=NULL);

# Have a way to modify data in the cache.
public function modify_cache($key,$object_id,$rights);

# Have a way to delete items from the cache.
public function delete_from_cache($key,$object_id,$rights=NULL);
```

ApplicationACL implements an interface for custom caching.

The different methods for caching require the implementation of an interface. But this allows for the greatest amount of control.

The interface was implemented because different organizations use different caching engines. Digg uses memcache, while for my blog I might use sessions, for example. The interface allows us to standardize how we save and request information.

```
define('CACHING_ON', '1'); // Are we caching the results? (bool int)
define('AUTO_CREATE', '1'); // Automatically create missing tables? (
?>
```

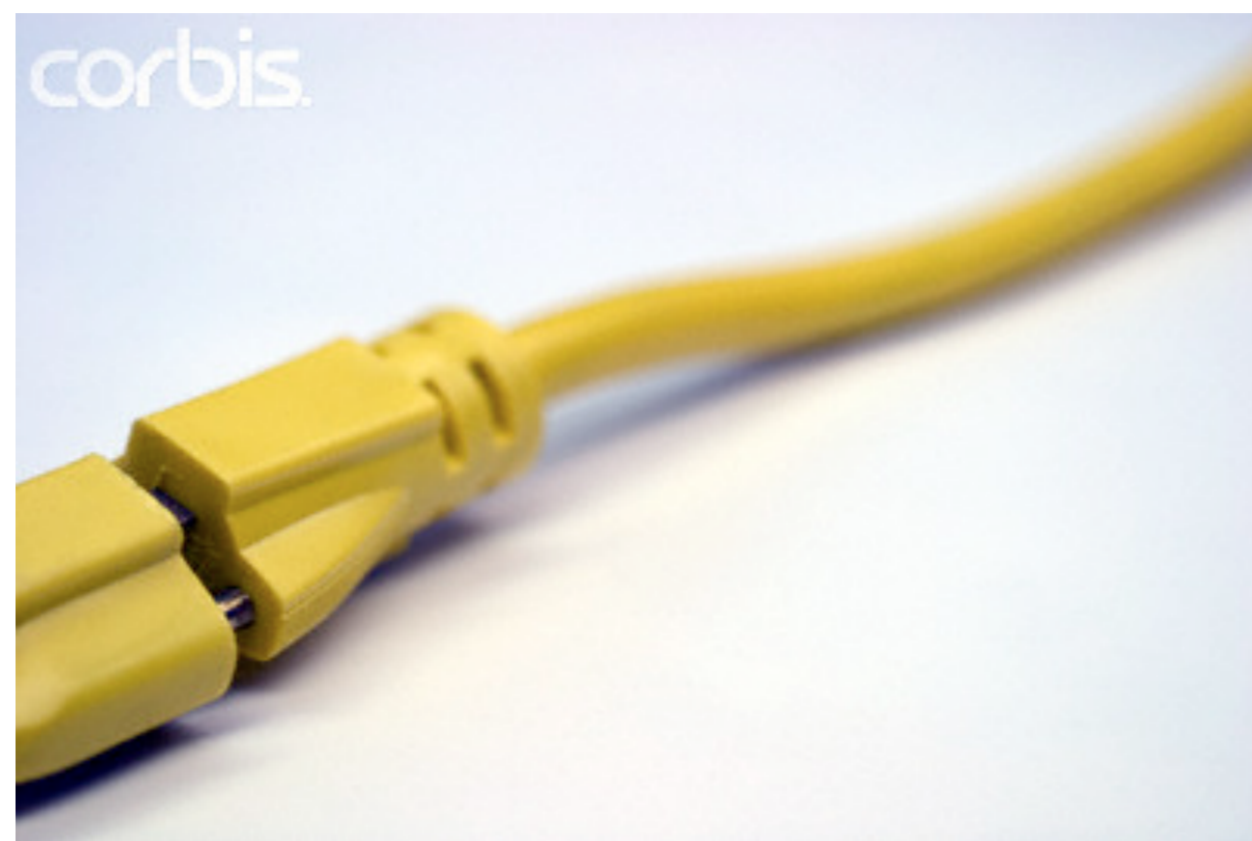
Caching is optional; ApplicationACL works either way.

It's worth noting that caching the permissions is optional. ApplicationACL works either way.

Sometimes you might not want to cache permissions. For example, internal applications that don't have high traffic might not place a considerable strain on the database server when making requests.

Today's examples were done with caching turned off.

The
controller
can be
extended to
manage
custom
permissions.



ApplicationACL doesn't live in a box. Though it's central controller is designed to remain unmodified, you are free to extend it any way you like.

ApplicationACL is in use by Greenpeace for their internal application. Let's take a look at their implementation.

in. However, you can override this and grant more or fewer permissions using the tables below.

Since this system is hierarchical, granting a higher level of permissions automatically grants permissions to all lower levels (e.g. Create implies Write and Read).

Permissions set for users here will override their group settings, so be careful! If you are not, you may end up granting more or fewer permissions than a user needs.

Users must log out and log back in before changes will take effect.

Manage Application Permissions:	Add...				
	None	Read	Write	Create	Delete
Menu: User Access (Remove)	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Menu: Bulk Email (Remove)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Menu: root (Remove)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>

Manage Record Permissions:	Add...				
	None	Read	Write	Create	Delete

Greenpeace uses hierarchical permissions.

Note that Greenpeace is using hierarchical permissions, and also application and record-based permissions.

The permissions controller can be extended to manage each one of these. In this case, the controller is intact but a custom permissions class has been written.

This makes ApplicationACL exceptionally powerful.
(TRANSITION TO NEXT SLIDE HERE)

**This control allows for
custom implementation.**

Contact Me:

Brandon Savage

brandon@

brandonsavage.net

Questions?

Twitter:

brandonsavage

AIM:

brandonsavage020

www.brandonsavage.net

I want to take a moment and thank everyone for coming out. My contact information is up on the screen. Does anyone have any questions?