

Implementing Access Control with Zend Framework

Darby Felton

PHP Developer, Zend Technologies



ZEND/PHP CONFERENCE & EXPO 8-11 OCTOBER 2007

Topics Overview

- Introduction to Zend Framework
- Authenticating with Zend_Auth
- Access Control Lists with Zend_Acl
- Putting it Together with MVC
- Example Application
- Q & A



Introduction to Zend Framework

Zend Framework facilitates development of PHP applications that require authentication and access control by providing flexible and extensible components built using the object-oriented features of PHP 5



Introduction to Zend Framework

- Designed to facilitate building web applications and web services with object-oriented PHP 5
- Open Source
 - § New BSD license is business-friendly
 - § Free for development and distribution
 - § CLA process assures that the code is free of legal issues



Introduction to Zend Framework

- “Extreme simplicity”
- Use-at-will architecture
- Designed for extensibility
- Extensive documentation and testing
- Continuous community involvement



Introduction to Zend Framework

- Class Library – over 150,000 lines of code
- Documentation – over 500 pages
- Quality & Testing – over 4,200 unit tests
- Over 2,000,000 downloads
- Supports PHP 5.1.4 and later



Introduction to Zend Framework

Special thanks to:

- Simon Mundy for the first production version of Zend_Acl and more
- Bryce Lohr for Zend_Auth_Adapter_Http and other contributions
- Ralph Schindler for his work on both Zend_Auth and Zend_Acl, including Zend_Auth_Adapter_DbTable
- The Zend Framework community for their invaluable feedback from applied use cases



Authenticating with Zend_Auth

Authentication – determining whether an entity is actually what it purports to be, based on some set of credentials



Authenticating with Zend_Auth

- Designed to authenticate the requester's identity against some authentication mechanism (e.g., HTTP Basic/Digest, database table, LDAP)
- Supports user-defined authentication adapters
- Available automatic identity persistence
- Configurable identity storage implementation
- Provides a simple authentication interface



Authenticating with Zend_Auth

- Zend_Auth adapters implement **Zend_Auth_Adapter_Interface**:

```
class MyAuthAdapter implements Zend_Auth_Adapter_Interface
{
    /**
     * Performs an authentication attempt
     * @throws Zend_Auth_Adapter_Exception
     * @return Zend_Auth_Result
     */
    public function authenticate()
    {
    }
}
```

Authenticating with Zend_Auth

When does `authenticate()` throw an exception?

- If and only if the authentication query cannot be answered
 - § Authentication service (e.g., DB, LDAP) is unavailable
 - § Cannot open password file
- Not under normal authentication failure circumstances
 - § Username does not exist in the system
 - § Password is incorrect

Authenticating with Zend_Auth

Authentication results are returned as a `Zend_Auth_Result` object, which provides:

- `boolean isValid()`
- `integer getCode()`
- `mixed getIdentity()`
- `array getMessages()`



Authenticating with Zend_Auth

- Two ways to authenticate against a Zend_Auth adapter:
 - § Indirectly, through `Zend_Auth::authenticate()`
 - § Directly, through the adapter's `authenticate()` method
- By indirect usage the authenticated identity is automatically saved to persistent storage (e.g., the PHP session)
- Direct usage of Zend_Auth adapters allows the user to decide what to do upon authentication



Authenticating with Zend_Auth

- Zend_Auth implements the Singleton pattern; exactly one instance of the Zend_Auth class is available at any time:

```
assert(Zend_Auth::getInstance() instanceof Zend_Auth);
```

- Exactly one request per PHP execution lifetime
- Operators `new` and `clone` are unavailable

Authenticating with Zend_Auth

- Zend_Auth automatically persists a successfully authenticated identity to the PHP session
- Override this behavior by passing an object that implements `Zend_Auth_Storage_Interface` to `Zend_Auth::setStorage()`
- If automatic identity storage is undesirable, developers may directly authenticate against a Zend_Auth adapter



Authenticating with Zend_Auth

- Using a Zend_Auth adapter indirectly:

```
$authAdapter = new MyAuthAdapter($username, $password);  
$auth = Zend_Auth::getInstance();  
$result = $auth->authenticate($authAdapter);  
if (!$result->isValid()) {  
    foreach ($result->getMessages() as $message) {  
        echo "$message\n";  
    }  
}
```

- Authenticated identity is saved automatically

Authenticating with Zend_Auth

- Using a Zend_Auth adapter directly:

```
$authAdapter = new MyAuthAdapter($username, $password);  
$result = $authAdapter->authenticate();  
if (!$result->isValid()) {  
    foreach ($result->getMessages() as $message) {  
        echo "$message\n";  
    }  
}
```

- No automatic storage of authenticated identity

Authenticating with Zend_Auth

Other useful Zend_Auth methods:

- `boolean hasIdentity()`
- `mixed getIdentity()`
- `void clearIdentity()`



Access Control Lists with Zend_Acl

Zend_Acl provides role-based access control lists functionality and privileges management



Access Control Lists with Zend_Acl

- Object-oriented design supports controlling access to certain protected objects by other requesting objects
- Complete PHP implementation
- Persistence does not require any backend technology; instances are serializable



Access Control Lists with Zend_Acl

- Definitions
 - § A Resource is an object to which access is controlled
 - § A Role is an object that may request access to a Resource
- Roles request access to Resources
 - § e.g., Person requests access to Car
- Roles and Resources must be added to the ACL before applying any rules upon them or querying against them
- Specify rules with `allow()` and `deny()`
- Query the ACL with `isAllowed()`



Access Control Lists with Zend_Acl

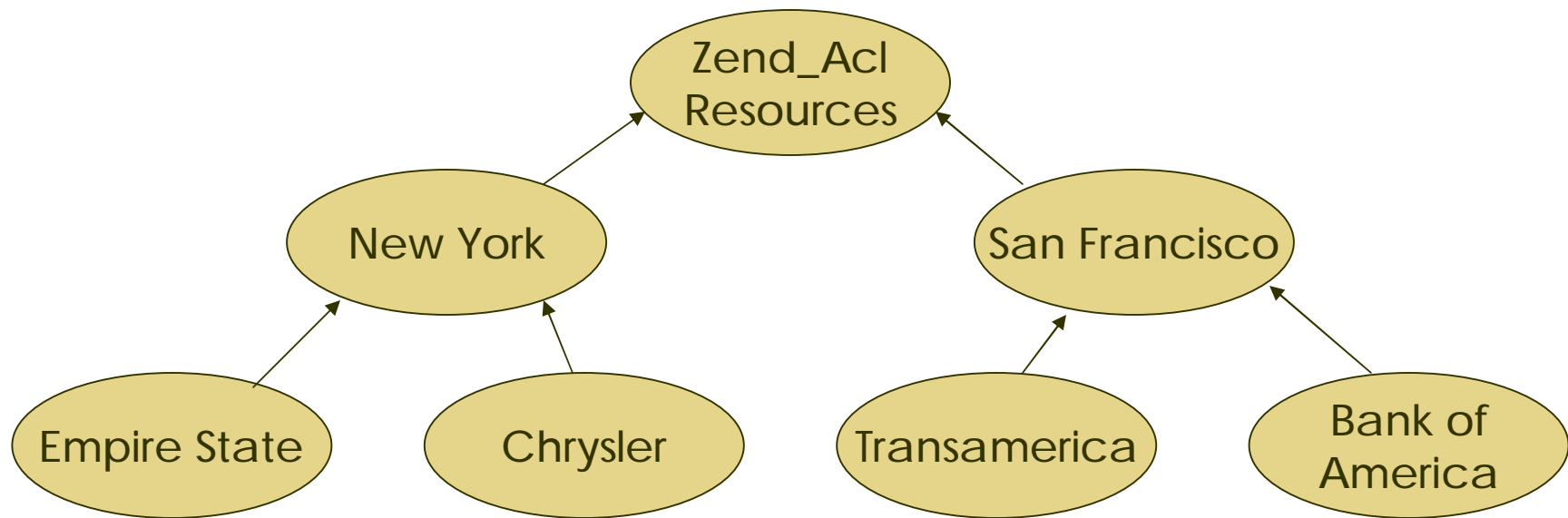
- Resource objects must implement **Zend_Acl_Resource_Interface**:

```
class MyResource implements Zend_Acl_Resource_Interface
{
    /**
     * @return string
     */
    public function getResourceId()
    {}
}
```

- Zend_Acl includes **Zend_Acl_Resource**

Access Control Lists with Zend_Acl

- Resources may be organized into a hierarchy:



- Rules are inherited from parent resources

Access Control Lists with Zend_Acl

- Example inheritance between resources:

```
<?php
$acl = new Zend_Acl();

$acl->addRole(new Zend_Acl_Role('guest'));

$acl->add(new Zend_Acl_Resource('New York'))
    ->add(new Zend_Acl_Resource('Empire State'), 'New York')
    ->add(new Zend_Acl_Resource('Chrysler'), 'New York');

$acl->allow('guest', 'New York')
    ->deny('guest', 'Empire State');

echo $acl->isAllowed('guest', 'Empire State') ?
    'allowed' : 'denied';
echo $acl->isAllowed('guest', 'Chrysler') ?
    'allowed' : 'denied';
```



Access Control Lists with Zend_Acl

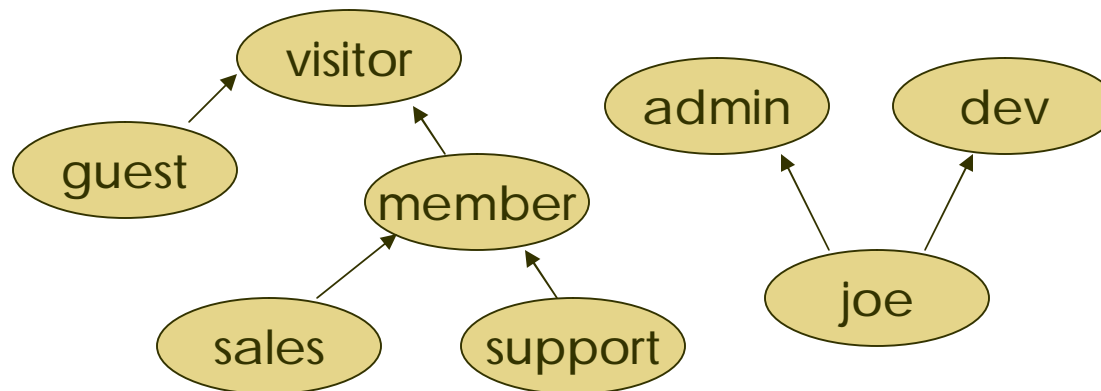
- Role objects must implement **Zend_Acl_Role_Interface**:

```
class MyRole implements Zend_Acl_Role_Interface
{
    /**
     * @return string
     */
    public function getRoleId()
    {}
}
```

- Zend_Acl includes **Zend_Acl_Role**

Access Control Lists with Zend_Acl

- Roles may be organized into a directed acyclic graph (DAG):



- Access control rules are inherited from parent roles
- Multiple inheritance ambiguity resolution

Access Control Lists with Zend_Acl

- Example role inheritance ambiguity resolution:

```
$acl = new Zend_Acl();

$acl->addRole(new Zend_Acl_Role('guest'))
    ->addRole(new Zend_Acl_Role('member'))
    ->addRole(new Zend_Acl_Role('admin'));

$parents = array('guest', 'member', 'admin');
$acl->addRole(new Zend_Acl_Role('someUser'), $parents);

$acl->add(new Zend_Acl_Resource('someResource'));

$acl->deny('guest', 'someResource');
$acl->allow('member', 'someResource');

echo $acl->isAllowed('someUser', 'someResource') ?
    'allowed' : 'denied';
```



Access Control Lists with Zend_Acl

- Supports "privileges" upon resources (e.g., "view" privilege upon an "article" resource)
- Usage is completely optional
- Privileges are string identifiers, not objects
- Privileges are specified with allow/deny rules

```
$acl->allow($someRole, $someResource, 'view');  
$acl->deny($someRole, $someResource, array('edit', 'delete'));  
if ($acl->isAllowed($someRole, $someResource, 'view')) {  
    ...  
}
```



Access Control Lists with Zend_Acl

- Rules may be removed from the ACL using `removeAllow()` and `removeDeny()`
- Specify the role(s), resource(s), and privilege(s) to which the removed rule must no longer apply

```
$acl = new Zend_Acl();  
$acl->allow(null, null, array('privilege 1', 'privilege 2'));  
assert(!$acl->isAllowed());  
assert($acl->isAllowed(null, null, 'privilege 1'));  
assert($acl->isAllowed(null, null, 'privilege 2'));  
$acl->removeAllow(null, null, 'privilege 1');  
assert(!$acl->isAllowed(null, null, 'privilege 1'));  
assert($acl->isAllowed(null, null, 'privilege 2'));
```



Access Control Lists with Zend_Acl

- Assertions provide support for conditional rules
- Examples:
 - § Allow between 8:00am and 5:00pm
 - § Deny from specific IPs or networks
 - § Allow only the author to edit an article
- Pass an instance of **Zend_Acl_Assert_Interface** to **allow()/deny()**
- The rule applies if and only if **assert()** returns **true**



Access Control Lists with Zend_Acl

- An example assertion class for screening requests from abusive IP addresses:

```
class My_Acl_Assert_DirtyIP implements Zend_Acl_Assert_Interface
{
    public function assert(Zend_Acl $acl,
                          Zend_Acl_Role_Interface $role = null,
                          Zend_Acl_Resource_Interface $resource = null,
                          $privilege = null)
    {
        return $this->_isDirtyIP($_SERVER['REMOTE_ADDR']);
    }

    protected function _isDirtyIP($address)
    {}
}
```



Access Control Lists with Zend_Acl

- Using a "DirtyIP" assertion object to deny access to known abusive IP addresses:

```
$acl = new Zend_Acl();  
$acl->deny(null, null, null, new My_Acl_Assert_DirtyIP());
```

- If the requesting IP is blacklisted (or not on a whitelist), then `assert()` returns `true`, and the deny rule applies, resulting in access denied

Putting it Together with MVC

The Model View Controller pattern separates an application design into three distinct roles, facilitating development and maintenance



Putting it Together with MVC

- Zend Framework provides implementations of the Front Controller and Model-View-Controller (MVC) patterns
- Neither Zend_Auth nor Zend_Acl requires use of these patterns, but it is helpful to see how to integrate authentication and access control rules with the Zend Framework controller systems
- TIMTOWTDI, so we present an example



Putting it Together with MVC

- Configure the Front Controller
 - § Set action controllers directory
 - § Register plug-ins (e.g., authentication and authorization)
 - § Set other options (e.g., throwing exceptions, returning the response)
- Set up the ACL for action controllers
- Set up any custom routes
- Dispatch the Front Controller



Example Application

"Example isn't another way to teach, it is the only way to teach" - Albert Einstein



Example Application

- Home page has personalized greeting
- Login form
- Logout feature
- User accounts stored in MySQL
- Users can change their own passwords and display names



Example Application

- Create the database and user account storage table:

```
CREATE DATABASE `myapp` ;
CREATE TABLE `myapp`.`user` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `username` char(32) NOT NULL,
  `password` char(32) NOT NULL,
  `fullname` char(32) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `username` (`username`)
) COMMENT='user accounts';
```

Example Application

- Add an administrative user and a regular user to the new table:

```
INSERT INTO user (username, password, fullname)
VALUES ('admin', MD5('admin'), 'Administrator'),
('someuser', MD5('someuser'), 'Some User');
```

Example Application

- The application will route all requests that do not correspond to an existing file or directory to the PHP application
- Using Apache and mod_rewrite, the following **.htaccess** file would work as desired:

```
RewriteEngine on
RewriteCond %{SCRIPT_FILENAME} !-f
RewriteCond %{SCRIPT_FILENAME} !-d
RewriteRule ^(.*)$ index.php/$1
```

Example Application

- The `index.php` file contains simply the following:

```
<?php  
  
require_once 'application/library/My/App.php';  
  
My_App::getInstance()->run();
```

- The application class will take care of setting up the Front Controller and dispatching the request



Example Application

- The interesting parts of `My_App::getInstance()->run()`:

```
$frontController = Zend_Controller_Front::getInstance();  
$frontController->throwExceptions(true)  
    ->registerPlugin(new My_Controller_Plugin_Auth())  
    ->returnResponse(true);
```

- Continue to set up the ACL...



Example Application

- Add resources and roles, and create the rules needed to enforce the application requirements

```
$acl = $this->getAcl();

$acl->add(new Zend_Acl_Resource('index'))
    ->add(new Zend_Acl_Resource('login'))
    ->add(new Zend_Acl_Resource('logout'))
    ->add(new Zend_Acl_Resource('profile'))

    ->addRole(new Zend_Acl_Role('anonymous'))
    ->addRole(new Zend_Acl_Role('member'), 'anonymous')
    ->addRole(new Zend_Acl_Role('admin'), 'member')

    ->allow()
    ->deny(null, 'profile')
    ->allow('member', 'profile');
```



Example Application

- Now, dispatch the request and catch any resulting exception:

```
try {  
    $response = $frontController->dispatch();  
    $response->sendResponse();  
} catch (Exception $e) {  
    echo $e->getMessage();  
}
```

- For this example, the exception is simply printed, but production applications should log the occurrence of an exception (see `Zend_Log`)

Example Application

- The Front Controller has an Auth plugin:

```
class My_Controller_Plugin_Auth extends Zend_Controller_Plugin_Abstract
{
    public function preDispatch(Zend_Controller_Request_Abstract $request)
    {
        $auth = Zend_Auth::getInstance();

        if ($auth->hasIdentity()) {
            switch ($auth->getIdentity()->username) {
                case 'admin':
                    $role = 'admin';
                    break;
                default:
                    $role = 'member';
                    break;
            }
        } else {
            $role = 'anonymous';
        } // continued on next slide...
    }
}
```



Example Application

```
// ...continued from previous slide
$request = $this->getRequest();

$controllerName = $request->getControllerName();

$acl = My_App::getInstance()->getAcl();

if (!$acl->has($controllerName)) {
    throw new Exception('Sorry, the requested controller does not'
        . ' exist as an ACL resource');
}

if (!$acl->isAllowed($role, $controllerName,
    $request->getActionName())) {
    $request->setControllerName('index')
        ->setActionName('denied')
        ->setDispatched(false);
}
}
}
```



Example Application

- The application controllers extend the following class:

```
class My_Controller_Action extends Zend_Controller_Action {
    public function preDispatch()
    {
        $view =
        Zend_Controller_Action_HelperBroker::getStaticHelper('viewRendererer')->view;

        $auth = Zend_Auth::getInstance();
        if ($view->authenticated = $auth->hasIdentity()) {
            $view->user = new My_Model_User($auth->getIdentity());
        } else {
            $view->user = new My_Model_User();
        }

        $view->baseUrl = Zend_Controller_Front::getInstance()->getBaseUrl();
    }

    public function __call($name, $args)
    { throw new Exception('Sorry, the requested action does not exist'); }
}
```



Example Application

- Home page (index/index)
 - § Personalized greeting
 - § Layout view script prints date and time
 - § Shows "Edit Profile" link only to authenticated users



Example Application

- Login function (login/index)
 - § Requires username of between 3 and 32 alphabetic characters
 - § Requires password of at least 5 characters
 - § Recalls the most recent reasons for login failure
 - § POSTs to login/process, which redirects
 - § Authenticated users do not see login form
- Logout feature (logout/index)



Example Application

- Edit Profile (profile/edit)
 - § Only authenticated members are authorized due to ACL rule
 - § Full name must be between 3 and 32 characters
 - § Password must be at least 5 characters
 - § Password must match verification field
 - § Recalls the most recent reasons for failure to save profile data
 - § POSTs to profile/edit/process, which redirects to profile/edit



Thank you!

More about Zend Framework:
<http://framework.zend.com>

