Advanced Authorization in DotNetNuke

Brandon Haynes
Hi, I’m Brandon.
Research Interests
Professional Interests
DotNetNuke Activities

- Core Team Member (Security)
- Framework and Module Security Guidance
- Current Research
  - Multi-Factor Authentication
  - Model Adaptation
  - Cloud Integration
Our agenda.

1. What is this all about?
2. How does this relate to DotNetNuke?
3. Show me something concrete, already.
You promised me authorization. Doesn’t that mean passwords and fingerprints and stuff?
Principal
Something that can be authenticated.

Identifier
Unique value used to reference a principal.
Thus, Authentication
(Matching principals to factors)

Knowledge  Inherence  Possession
We’re here to talk about what happens after authentication: Authorization.

(Shameless Plug: If you’re interested in this stuff, be sure to attend Open Force 2009, where I will be talking about module security in much greater detail!)
Authorization

• I know who you are.
• But what can you do?
  – Or access?
• Involves two things:
  – Policy definition (who can do what)
  – Policy enforcement (no, you can’t do that)
Act 2

What does this have to do with DotNetNuke, anyway?
Pre-5.1 Authorization: Architecture

- ModulePermissionController
- TabPermissionController
- FolderPermissionController
- Module 1
- Module 2
- Module 3

Internal Classes

Core Framework

DotNetNuke.dll
Pre-5.1 Authorization: Classes

### ModulePermissionController

**Class**

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>- HasModulePermission (+ 2 overloads)</td>
</tr>
<tr>
<td>- AddModulePermission (+ 1 overload)</td>
</tr>
<tr>
<td>- DeleteModulePermission</td>
</tr>
<tr>
<td>- DeleteModulePermissionsByModuleID</td>
</tr>
<tr>
<td>- DeleteModulePermissionsByUserID</td>
</tr>
<tr>
<td>- GetModulePermission</td>
</tr>
<tr>
<td>- GetModulePermissionsByModuleID</td>
</tr>
<tr>
<td>- GetModulePermissionsByPortal</td>
</tr>
<tr>
<td>- GetModulePermissionsByTabID</td>
</tr>
<tr>
<td>- GetModulePermissionsCollectionByModuleID (+ 2 overloads)</td>
</tr>
<tr>
<td>- GetRoleNamesFromRoleIDs</td>
</tr>
<tr>
<td>- UpdateModulePermission</td>
</tr>
</tbody>
</table>
Pre-5.1 Authorization: Methods

ModulePermissionController
Class

Methods

HasModulePermission (+ 2 overloads)

Public Shared Function HasModulePermission (_
    ByVal permissions As ModulePermissionCollection, _
    ByVal key As String) As Boolean
    Return PortalSecurity.IsInRoles(permissions.ToString(key))
End Function
Okay, so what’s new?
5.1 CE Authorization Provider

Core Framework

- ModulePermissionController
  - Class

- TabPermissionController
  - Class

- FolderPermissionController
  - Class

Authorization

DotNetNuke.dll

CorePermissionProvider
- Class
- `Instance() : PermissionProvider`

DotNetNuke.Provider.CorePermissionProvider.dll
## 5.1 CE CorePermissionProvider

<table>
<thead>
<tr>
<th>Category</th>
<th>Overridable Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Module Permissions</strong></td>
<td></td>
</tr>
<tr>
<td>CanAdminModule</td>
<td>CanImportModule</td>
</tr>
<tr>
<td>CanDeleteModule</td>
<td>CanManageModule</td>
</tr>
<tr>
<td>CanEditModuleContent</td>
<td>CanManageModule</td>
</tr>
<tr>
<td>CanExportModule</td>
<td>HasModulePermission</td>
</tr>
<tr>
<td>GetModulePermissions</td>
<td></td>
</tr>
<tr>
<td>DeleteModulePermissionsByUser</td>
<td></td>
</tr>
<tr>
<td><strong>Page Permissions</strong></td>
<td></td>
</tr>
<tr>
<td>CanAddContentToPage</td>
<td>CanExportPage</td>
</tr>
<tr>
<td>CanAddPage</td>
<td>CanImportPage</td>
</tr>
<tr>
<td>CanAdminPage</td>
<td>CanManagePage</td>
</tr>
<tr>
<td>CanCopyPage</td>
<td>CanManagePage</td>
</tr>
<tr>
<td>CanDeletePage</td>
<td>CanManagePage</td>
</tr>
<tr>
<td>DeleteTabPermissionsByUser</td>
<td></td>
</tr>
<tr>
<td><strong>Folder Permissions</strong></td>
<td></td>
</tr>
<tr>
<td>CanAdminFolder</td>
<td>CanCopyFolder</td>
</tr>
<tr>
<td>CanAddFolder</td>
<td>CanCopyFolder</td>
</tr>
<tr>
<td>CanCopyFolder</td>
<td>CanCopyFolder</td>
</tr>
<tr>
<td>HasFolderPermission</td>
<td>SaveFolderPermissions</td>
</tr>
<tr>
<td>DeleteFolderPermissionsByUser</td>
<td></td>
</tr>
<tr>
<td>GetFolderPermissionsCollectionByFolder</td>
<td></td>
</tr>
</tbody>
</table>

**ATTACK OF THE GIANT TABLE!**
## 5.1 CE CorePermissionProvider

<table>
<thead>
<tr>
<th>Category</th>
<th>Overridable Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module Permissions</td>
<td>CanViewModule</td>
</tr>
<tr>
<td></td>
<td>CanDeleteModule</td>
</tr>
<tr>
<td></td>
<td>CanAdminModule</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Page Permissions</td>
<td>CanViewPage</td>
</tr>
<tr>
<td></td>
<td>CanAddPage</td>
</tr>
<tr>
<td></td>
<td>CanDeletePage</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Folder Permissions</td>
<td>CanViewFolder</td>
</tr>
<tr>
<td></td>
<td>CanAddFolder</td>
</tr>
<tr>
<td></td>
<td>CanDeleteFolder</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
If TabPermissionController.CanAddPage Then
    Dim controller As New TabController
    controller.AddTab(...) 
End If

5.1 Authorization: The Big Picture

Module (e.g., Control Panel)

Internal Classes (e.g., TabController)

Core Framework

ModulePermissionController Class

TabPermissionController Class

FolderPermissionController Class

PermissionProvider Class

public

CorePermissionProvider Class

Authorization

AddTab
Act 3

Show me how to use it.
Hello World Demo

– MagicUsernamePermissionProvider
– Gives Brandon page admin rights across all portals
– About the simplest PermissionProvider possible
Hello World: Configuration

<configuration>

...

<dotnetnuke>

...

<permissions
defaultProvider="MagicNameAuthorizationProvider">
    <providers>
        <clear/>
        <add name="MagicNameAuthorizationProvider"
             providerPath="~..." />
    </providers>
</permissions>
</dotnetnuke>
</configuration>
class MagicUsernamePermissionProvider
    : PermissionProvider
{
    Provider Settings

    public override bool CanAdminPage(TabInfo objTab)
    {
        if (UserController
            .GetCurrentUserInfo()
            .Username == "Brandon")
            return true;
        else
            return base.CanAdminPage(objTab);
    }
}
Now for some fun.
Amazon Simple Storage Service (S3)

“Amazon S3 is storage for the Internet. It is designed to make web-scale computing easier for developers.”
Amazon Simple Storage Service (S3)

“Amazon S3 is intentionally built with a minimal feature set.”

- Accounts
- Buckets
- Abstract objects
- Simple Permissions
- Very simple web service API
  - Access via shared secret
DotNetNuke S3 Integration

• Data Provider
• Authorization Provider
S3 Provider Architecture: Data

- **DataProvider**
  - MustInherit Class

- **SqlDataProvider**
  - Class
    - Inheriting from DataProvider

- **AmazonS3DataProvider**
  - Class
    - Inheriting from SqlDataProvider
      - Methods:
        - GetFiles
        - GetFolder (+ 1 overload)
        - GetFoldersByPortal
        - ...

- **CompositeDataReader**
  - Class
    - Inheriting from IDataReader

- Amazon Web Services

- SQL Server
public override IDataReader GetFoldersByPortal(int PortalID)
{
    return base.GetFoldersByPortal(PortalID)
        .Union(S3Service.GetAllBuckets());
}
S3 Architecture: Authorization

**PermissionProvider**
Class

**AmazonS3AuthorizationProvider**
Class
- PermissionProvider

**Methods**
- CanAddFolder
- CanAdminFolder
- CanCopyFolder
- CanDeleteFolder
- CanManageFolder
- CanViewFolder
public override bool CanViewFolder(FolderInfo folder)
{
    var user = UserController.GetCurrentUserInfo();
    bool isAuthorized;

    if (!S3Folder.IsS3Folder(folder.FolderID))
        isAuthorized = base.CanViewFolder(folder);
    else
        isAuthorized = (user.S3Service()
            .QueryBucket(folder.ToBucketName()) ==
            LitS3.BucketAccess.Accessible);

    return isAuthorized;
}
DotNetNuke and S3: Discussion

- File-based permissions
- CRUD
- Shared secrets
- Performance
- Available at brandonhaynes.org.
Points to Remember

• Authorization is a first-class extension point as of version 5.1.
• It is extended via the provider pattern, just like any other DotNetNuke provider would be
• The provider deals with page-, module-, and folder-related authorizations.
• Custom authorization allows for framework extension to interesting and novel scenarios.
Thanks to:
Will Strohl, Darrell Hardy, Ryan Morgan, Stan Schultes and all the DoDNN Organizers.

All of the DotNetNuke Core Team, Corporation, and Community.

And all the volunteers, sponsors, and participants that made this event possible.
Insert questions here.

More questions? Contact me via brandonhaynes.org.