

Designing Application Managed Authorization

Dominick Baier (dbaier@ernw.de)
.NET Security Consultant / BS 7799 Lead Auditor
ERNW GmbH



Outline

- **.NET Framework Authorization Mechanisms**
- **Identities and Principals**
- **Using Role Based Security and Access Checks**
- **Authorization in Enterprise Applications**
- **Microsoft Authorization Manager**
- **Extending .NET Role Based Security**

developmentor



What is?

- **Authentication**
 - The process of identifying a principal
 - A principal can be a user, a computer, a network device or an assembly
 - The principal is identified through credentials
- **Authorization**
 - Is the confirmation that an authenticated principal has permissions to perform a specific operation
 - Protection allows only appointed users to perform certain actions, and it prevents malicious acts

developmentor



Performing Authorization on each Layer

User Interface

How do I customize the user interface based on a user's role?
How do I change the state of the UI based on the user's permissions?

Business Logic

How can I write code to manage authorization in business processes?
How do I make sure I know what identity to authorize on in server environments?

Data Access

How do I prevent sensitive data from reaching unauthorized users?
How do I ensure anonymous users don't modify data?



Authorization Mechanisms

- **System Authorization**
 - NTFS, Active Directory, SQL Server, MSMQ ACLs
- **.NET Code Access Security Authorization**
 - Perform authorization based on the code's origin
 - Sandbox your code to prevent misuse
- **Application Authorization**
 - The application code itself authorizes user actions
 - "Is the user allowed to place an order exceeding 5000\$?"

developmentor



Use Roles for Authorization

- **A Role is a category or a set of users who share the same security privileges**
 - Easier to use than authorizing individual users
 - Memberships can change without having to adjust the logic
- **Roles based on the business organization**
 - Manager
 - Employee
 - Customer
- **Roles based on the types of operations**
 - CanDeleteCustomer
 - CanApproveClaim
 - CanShutDownNuclearReactor

developmentor



Using Role-Based Security in .NET Applications

- .NET provides two interfaces for Role-Based Security in the System.Security.Principal Namespace

```
interface IIdentity {  
    bool IsAuthenticated { get; }  
    string AuthenticationType { get; }  
    string Name { get; }  
}
```

```
interface IPrincipal {  
    IIdentity Identity { get; }  
    bool IsInRole(string roleName);  
}
```



Identities

- The .NET Framework provides four classes that implement the `IIdentity` interface
 - `WindowsIdentity`
 - `GenericIdentity`
 - `PassportIdentity`
 - `FormsIdentity`

```
WindowsIdentity currentIdentity =  
    WindowsIdentity.GetCurrent();
```

developmentor



8

Principals

- The IPrincipal interface links user roles and identities
- Linking the IPrincipal object to the thread provides easy access, and you don't have to carry it around
 - Plumbing does this (e.g. ASP.NET)
 - Or do it yourself (later...)

```
class Plumbing {  
    public static void DoHeavyLifting() {  
        IPrincipal client = AuthenticateUserSomehow();  
        Thread.CurrentPrincipal = client;  
        Application.ImplementBusinessLogic();  
    }  
}
```

developeror



Performing Role Check

- The `IPrincipal.IsInRole()` Method

```
class Application {  
    public static void ImplementBusinessLogic() {  
        if (Thread.CurrentPrincipal.IsInRole("Managers")) {  
            // do something privileged  
        }  
    }  
}
```

developer



10

The ability to work with `Identity` is a privileged operation. Grant the `SecurityPermissionAttribute.ControlPrincipal` to Applications.

The following methods require this permission

- `AppDomain.SetThreadPrincipalPolicy()`
- `WindowsIdentity.GetCurrent()`
- `WindowsIdentity.Impersonate()`
- `Thread.CurrentPrincipal()`

Manually changing `HttpContext.User` automatically updates the `Thread.CurrentPrincipal` for all threads executing within the same HTTP context. Changing `Thread.CurrentPrincipal` does not affect the `HttpContext.User` property. It affects only the chosen thread for the remainder of the request.

Performing Role Checks - Declaratively

- **PrincipalPermissionAttribute** uses **Thread.CurrentPrincipal**
 - allows you to add declarative checks against client identity
 - JIT compiler inserts the checks at the start of the method

```
using System.Security.Permissions;

[PrincipalPermission(SecurityAction.Demand, Authenticated=true)]
class PetStore {
    public void PetAnimals() { ... }
    public void BuyAnimals() { ... }

    [PrincipalPermission(SecurityAction.Demand, Role="Staff")]
    public void FeedAnimals() { ... }

    [PrincipalPermission(SecurityAction.Demand, Role="Managers")]
    public void GiveRaise() { ... }
}
```

developer



11

To call `PetAnimals` or `BuyAnimals`, the client must be authenticated.

To call `FeedAnimals`, the client must be authenticated AND in the Staff role.

To call `GiveRaise`, the client must be authenticated AND in the Managers role.

You can use `PERMVIEW.EXE` to find all declarative permissions like this:

```
permview /decl petstore.dll
```

Note that if you put two `PrincipalPermission` attributes next to each other (on the same method, say), you're creating an "or" expression, not an "and" as shown above, where the two attributes come from different places (a class attribute and a method attribute). Don't let this surprise you.

Performing Role Checks - Imperatively

- Basically the same as calling `IsInRole()`, but will generate a `SecurityException` if demand fails

```
try {  
    PrincipalPermission OperatorPermission =  
        new PrincipalPermission(null, "Operator");  
  
    OperatorPermission.Demand();  
}  
catch (SecurityException se) {}
```



Performing Role Checks - Imperatively

- Implementing an OR scenario

```
PrincipalPermission OperatorPermission =  
    new PrincipalPermission(null, "Operator");  
PrincipalPermission TechnicalStaffPermission =  
    new PrincipalPermission(null, "TechnicalStaff");  
  
(OperatorPermission.Union(TechnicalStaffPermission)).Demand();
```

- Implementing an AND scenario

```
OperatorPermission.Demand();  
TechnicalStaffPermission.Demand();
```



Thread.CurrentPrincipal and multiple Threads

- **What if I make an asynchronous call?**
 - BeginInvoke copies Thread.CurrentPrincipal to worker thread
 - Thread.Start copies Thread.CurrentPrincipal to new thread
- **Watch out for these special cases**
 - ThreadPool.QueueUserWorkItem doesn't copy it
 - System.Threading.Timer doesn't copy it

developer



14

Performing Authorization based on Windows Authentication

- **Windows uses NTLM or Kerberos to validate credentials**
- **A successful authentication produces a Windows token**
- **Link this Token to Thread.CurrentPrincipal**
 - Manual or with AppDomain.PrincipalPolicy
- **Windows Roles have the Format : AuthorityName\GroupName**
 - Watch out for localized names
 - Use the WindowsBuiltInRole enumeration
 - NT Authority\Network Service ==
NT Autorität\Netzwerk Dienst
 - Use Environment.UserName and Environment.MachineName to get these values at runtime

developer



15

Performing Authorization based on Windows Authentication

- Some variations

```
static void Main()
{
    AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);
    WindowsPrincipal principal = (WindowsPrincipal)Thread.CurrentPrincipal;

    Console.WriteLine("My name is {0}", principal.Identity.Name);
    if (principal.IsInRole(WindowsBuiltInRole.Administrator)) {}
    if (principal.IsInRole(WindowsBuiltInRole.User)) {}
    if (principal.IsInRole(@"LEASTPRIVILEGE\Domain Admins")) {}
}

[PrincipalPermission(SecurityAction.Demand,
                    Role=@"LEASTPRIVILEGE\Domain Admins")]
static void ShutDownNuclearReactor() {}
```

developeror



16

Performing Authorization by Using Application-Defined Roles

- To be independent from Windows Groups you can also use application defined roles
- Base the Identity on the Windows account and couple it with your own roles

```
GenericIdentity identity =  
    new GenericIdentity(WindowsIdentity.GetCurrent().Name);  
  
// normally you would look up the roles in a database or  
// config file  
Thread.CurrentPrincipal = new GenericPrincipal(identity,  
    new string[] { "Manager", "CanDeleteCustomer" } );
```



Perform Authorization Based on Non-Windows Accounts

- **You might not be able to use Windows authentication**
 - ASP.NET
 - Custom Authentication Systems (RSA SecurID...)
- **You have to do the authentication yourself**
 - Gather and verify credentials (typically against a database)
 - Construct an Identity
 - Couple roles with the Identity
 - Set Thread.CurrentPrincipal

developmentor



18

Perform Authorization Based on Non-Windows Accounts

- ASP.NET Forms Authentication

```
public void Application_AuthenticateRequest
(object sender, EventArgs e)
{
    string[] roles = getRoles(Context.User.Identity);

    Context.User = new GenericPrincipal(
        Context.User.Identity, roles);
}
```

developer



19

Directly after AuthenticateRequest fires an undocumented event called DefaultAuthentication. One hardwired module always registers for this event, its name is "DefaultAuthenticationModule". This module has only one job: copy HttpContext.User → Thread.CurrentPrincipal

Perform Authorization in an Enterprise Application

- **Most of today's applications are multi-layered**
 - User Interface Tier
 - Business Tier
 - Data Tier
- **You have to authorize in every single layer**
 - There are no trusted environments!
- **Perform authorization as early as possible in the process**
 - The earlier you can deny access the less resources are bound

developmentor



Performing Authorization in the User Interface Tier

- **Enable or disable or show and hide controls based on the user's role membership**
- **Change the flow from one form (or page) to the next**
- **Do not base the security of your system exclusively on application authorization within the user interface**
- **If the user interface limits the data the user can view, don't read unnecessary data to prevents leaks**

developmentor



21

Altering the User Interface

```
// Alter Controls
if (User.IsInRole("CanApproveClaims")) {
    cmdApprove.Visible = true;
    ClaimGrid.Columns[3].Visible = true;
}
```

```
// Alter Page Flow
if (User.IsInRole("GeneralManager"))
    Server.Transfer("ChooseDepartment.aspx");
else if (User.IsInRole("StoreManager"))
    Server.Transfer("ChooseEmployee");
```



Performing Authorization in the Business Tier

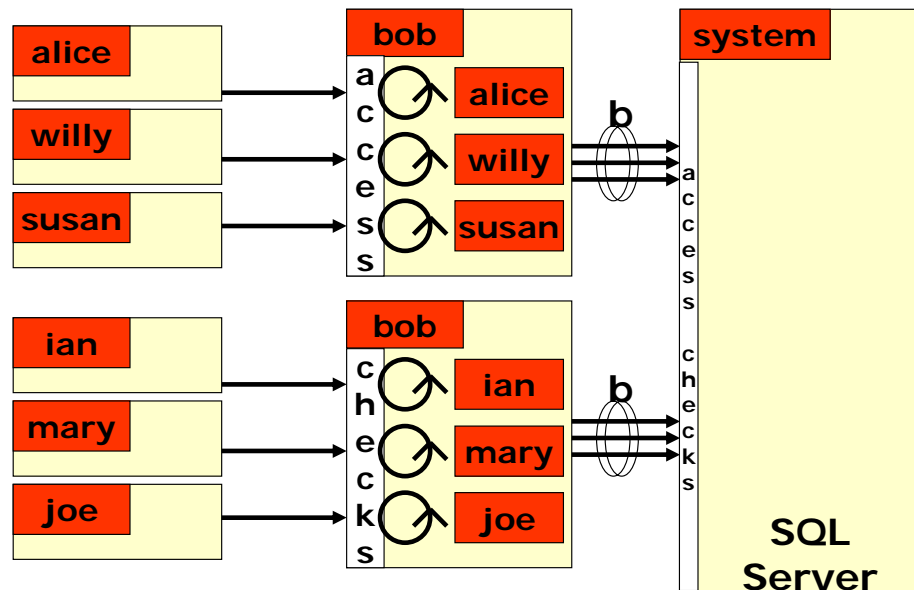
- **Authorize an operation that performs a business process**
- **Authorize a call from an external, nontrusted source before calling an internal business component**
- **Authorize calls to other parts of your distributed system or external services that are not part of your application**
- **Decide what kind of security model you want to use**
 - Impersonate/Delegate Model
 - Trusted Subsystem Design

developmentor



23

Trusted Subsystem



developmentor



24

Note that the middle tier account ("Bob") should still be as restricted as possible. It should only be granted access to the database(s) the middle tier needs to know about, and it should only be allowed to do things the middle tier should be able to do (for example, the middle tier very likely doesn't ever need to drop tables, or even delete records in some cases).

Remember the principal of least privilege!

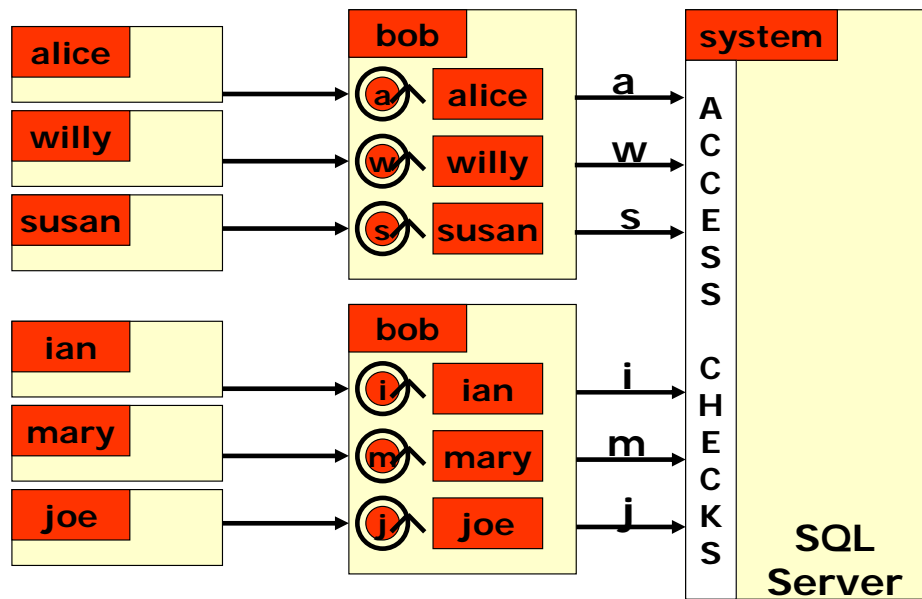
Benefits:

- scalability – unauthorized requests never make it to the back end + connection pooling
- middle tier doesn't have loads of client network credentials (no delegation required)

Drawbacks:

- must trust middle tier to do authorization checks
- may need to trust application developers to write custom authorization logic
- middle tier must supply the back end with client id for auditing purposes

Impersonation/Delegation



developer



25

We must warm up a new database connection for each request in this case, and SQL server does fine grained access checks against end users.

Benefits:

- authorization policy is kept close to the data

Drawbacks:

- not as scalable due to increased load on server
- if you use unconstrained delegation, this can be very dangerous

Programming Impersonation

```
using System.IO;
using System.Security.Principal;
using System.Security.Permissions;
using System.Threading;

[PrincipalPermission(SecurityAction.Demand, Authenticated=true)]
void DoSomeWorkForMyClient() {

    WindowsIdentity client = (WindowsIdentity)Thread.CurrentPrincipal.Identity;

    // temporarily impersonate the client to access a file
    WindowsImpersonationContext ctx = client.Impersonate();
    try {
        // open a file on the client's behalf
        using (Stream s = new FileStream("someFile.txt", FileMode.Open)) {
            // use the client's file...
        }
    }
    finally {
        ctx.Undo();
    }
}
```

developer



26

Impersonation is a privileged operation. Grant the "Impersonate after Authentication" privilege to the service account.

Performing Authorization in the Business Tier

- **There are several techniques you can use to do authorization in the business tier**
 - Depends on your design
 - Unfortunately you can't always easily mix them
- **.NET Role Based Security**
- **COM+ Role Based Security**
- **Microsoft Authorization Manager**



COM+ Role Based Security

- **The COM+ infrastructure doesn't expose the client's token through Thread.CurrentPrincipal**
 - You can't use .NET role based security with COM+
- **COM+ exposes the client information through the SecurityCallContext**
- **COM+ offers its own role system**

```
private string getCallerName()  
{  
    SecurityCallContext sc = SecurityCallContext.CurrentCall;  
    return sc.DirectCaller.AccountName;  
}
```

developmentor



28

Be aware that using the SecurityCallContext when security has been disabled (by an admin e.g.) generates an exception!

There's a paper on MSDN that shows you how to unify both COM+ and .NET role based security.

<http://msdn.microsoft.com/msdnmag/issues/02/05/rolesec/>

COM+ Roles



developer



29

Checking COM+ Roles

- If security is disabled, `IsCallerInRole()` will always return true!

```
void IStoreFront.AddContent(Content C)
{
    if (C.Area == "Executive" &&
        !ContextUtil.IsCallerInRole("ContentManager"))
        throw new SecurityException("...");
}
```



Impersonation in COM+

- This is also different...

```
void IServer.OpenFileInClientContext(string FileName)
{
    try {
        COMSec.CoImpersonateClient();
        // Do work on behalf of the client
    }
    finally {
        COMSec.CoRevertToSelf();
    }
}
```

developmentor



32

You have to **DLLImport** those calls.

```
class COMSec {
    [DllImport("OLE32.DLL", CharSet=CharSet.Auto)]
    public static extern uint CoImpersonateClient();
    [DllImport("OLE32.DLL", CharSet=CharSet.Auto)]
    public static extern uint CoRevertToSelf();
}
```

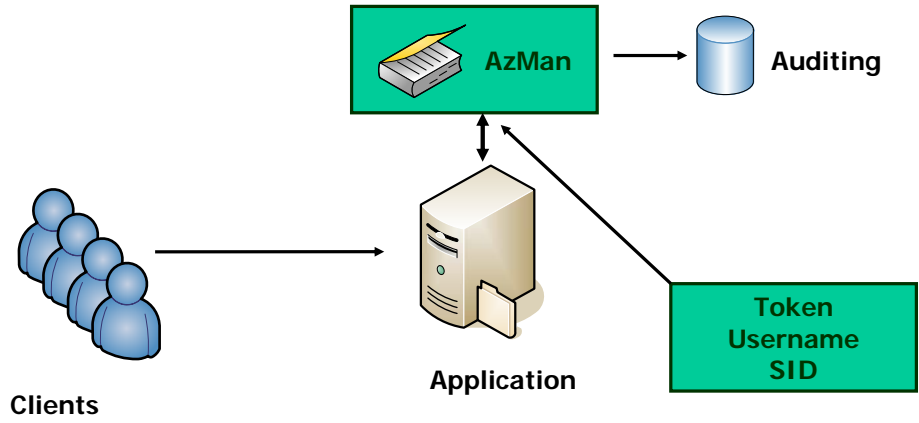
Microsoft Authorization Manager

- **AzMan is a COM library that performs role based access checks**
 - Looks similar to COM+ Roles but has more features
 - Is not tied to a specific technology
 - Further tries to separate application logic from the customers' application usage policies
- **New component in W2K3 but is now available down-level**
- **Role and access definitions can be stored in**
 - XML files
 - Active Directory/ADAM
- **When stored in Active Directory you also get auditing**

developmentor



Microsoft Authorization Manager

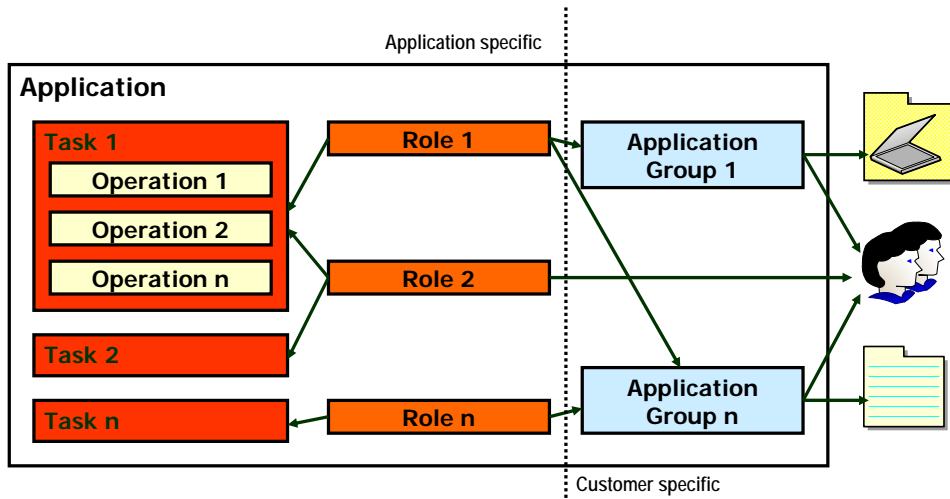


developmentor



34

Microsoft Authorization Manager

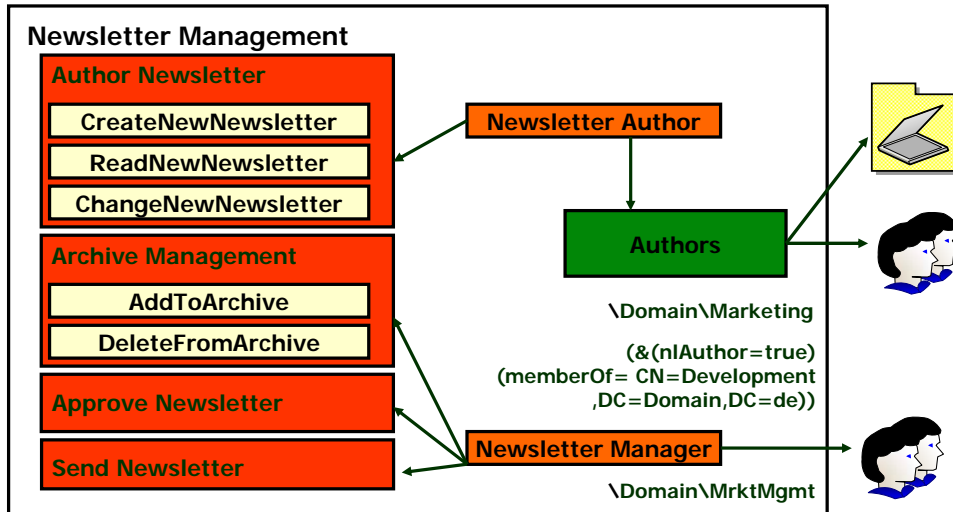


developer



35

Microsoft Authorization Manager



developeror



36

Microsoft Authorization Manager

```
AzAuthorizationStore store = new AzAuthorizationStoreClass();

// Opening an XML Store
store.Initialize(0, @"msxml://d:\etc\azNewsletter.xml",
    null);

// Opening an Active Directory Store
store.Initialize(0, @"msldap://CN=NewsletterApp,
    CN=Program Data,DC=LEASTPRIVILEGE,DC=LOCAL", null);

// Loading an AzMan Application
IAzApplication app = store.OpenApplication(
    "Newsletter Management", null);
```

developmentor



37

Microsoft Authorization Manager

```
// Initialize the client context with the current Windows Token
IAzClientContext ctx =
    app.InitializeClientContextFromToken(null, null);

// ...or grab the token from a WindowsIdentity
WindowsIdentity id = (WindowsIdentity)User.Identity;
IntPtr htok = id.Token;

IAzClientContext ctx =
    app.InitializeClientContextFromToken((ulong)htok, null);

// ... or use a username
IAzClientContext ctx =
    app.InitializeClientContextFromName(name, domain, null);
```

developer



38

Microsoft Authorization Manager

- After establishing a context you can retrieve the AzMan roles for the client

```
public string[] GetRoles(IAzClientContext ctx)
{
    object[] a = (object[])ctx.GetRoles("");
    string[] roles = new string[a.Length];
    Array.Copy(a, roles, a.Length);
    return roles;
}
```



Microsoft Authorization Manager

- or do access checks

```
public bool accessCheck(IAzClientContext ctx,
    int operationID)
{
    const int NO_ERROR = 0;

    object[] operations = { operationID };
    object[] scopes = { "" };
    object[] results = (object[])
        ctx.AccessCheck("Audit info",scopes,
            operations, null, null, null, null);
    int result = (int)results[0];
    if (NO_ERROR == result)
        return true;
    else
        return false;
}
```

developmentor



40

Microsoft Authorization Manager

- The AzMan engine is not bound to Windows accounts
- You can establish your own SID scheme
- This is currently not supported by the GUI – but the API is there
- **S-1-9-AppInstanceID-UserID**
 - S-1-9-1-1 (first app, first user)

developmentor



41

Microsoft Authorization Manager

- Adding SIDs to Application Groups

```
public void AddSidToGroup (string Sid, string ApplicationGroup)
{
    IAzApplicationGroup appGroup =
        getApplicationGroup(ApplicationGroup);

    appGroup.AddMember(Sid, null);
    appGroup.Submit(0, null);
}
```



Microsoft Authorization Manager

- Authenticate the user and get an AzMan Context

```
public IAzClientContext GetContext(string user, string password)
{
    string sid = AuthenticateUser(user, password);

    return app.InitializeClientContextFromStringSid(sid,1, null);
}
```

Don't check
against a Windows
Authority



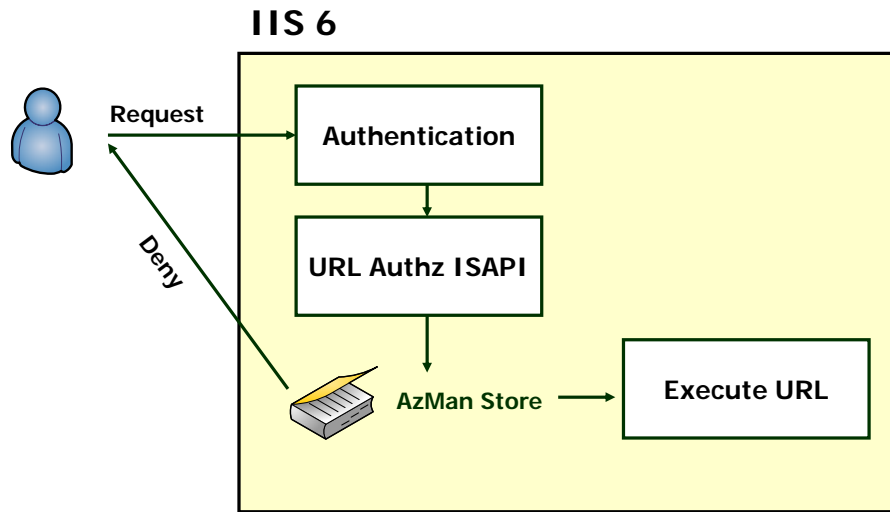
IIS 6 URL Authorization with AzMan

- **AzMan ISAPI Extension**
 - Maps virtual directories to AzMan Tasks
 - "Who is allowed to access which Web- or Smart-App" can be configured by policy
 - This policy can be stored centrally in Active Directory/ADAM
 - but also adds another layer of complexity

developmentor



IIS 6 URL Authorization



developmentor



45

Performing Authorization in the Data Tier

- Prevent sensitive data from propagating outside the data tier
- Prevent unauthorized modification of data
- Always limit access to data by using stored procedures or views rather than granting direct access to tables
- Use the security features of your DBMS to limit access
- Consider using different connection strings / accounts for different parts of your application
 - read / write only
 - read / write
 - integrated / sql server authentication

developmentor



46

Performing Authorization in the Data Tier

- Removing data based on role membership

```
public DataSet GetEmployeeData()  
{  
    // retrieve data  
    if (!Thread.CurrentPrincipal.IsInRole("Manager"))  
        data.Tables["HR"].Columns.Remove("Salary");  
  
    return data;  
}
```



Performing Authorization in the Data Tier

- ...or perform different work in stored procedures

```
create procedure GetEmployeeData
  @EmployeeName varchar(50),
  @IsManager bit
as
  if @IsManager = 1
    select * from Employees where EmployeeName = @EmployeeName
  else
    select FirstName, LastName from Employees
```



Performing Authorization in the Data Tier

- **Column based security is easy in databases**
 - Use the built in System Authorization
- **Row based security is harder**
 - When using impersonation the callers identity flows to SQL Server and is available via SUSER_SNAME
 - Can get complex with more advanced scenarios

```
create view EmployeeData
as
select * from Employees where EmployeeName = SUSER_SNAME()
```



Extending .NET Role Based Security

- You should create your own implementation of `IIdentity` and `IPrincipal` when you require more functionality
- `WindowsIdentity`, e.g. encapsulates Windows Tokens
- This helps to separate your business logic from your authorization logic
- Reusability

developmentor



50

Extending .NET Role Based Security

- **Guidelines**
 - Implement `IIdentity` and `IPrincipal`
 - Use lazy initialization and caching where possible
 - Attach your `IPrincipal` object to the executing thread
- **Create a custom identity when you need user-specific info**
- **Create a custom principal if you want to extend role-related permissions**

```
CustomPrincipal.IsInAnyRole (string[] roles);  
string[] roles = CustomPrincipal.Roles;  
bool hasAccess = AzManIdentity.AccessCheck("DeleteOp");
```



Summary

- .NET provides a rich framework for Authorization
- Use or extend `IPrincipal` and `Identity` to take advantage of declarative and imperative role checks
- Separate application and authorization Logic

- The technology you are using in your Middle-Tier affects your authorization
- Consider using Authorization Manager

- Authorize on each layer
- Think Defense in Depth



- **Questions?**

- **Get the Slides and Source Code at
<http://www.leastprivilege.com>**

developmentor



53

Links

- **Authorization Manager**

- <http://www.leastprivilege.com/PermaLink.aspx?guid=7cd21d34-8d8d-44ef-888e-153b2c272f91>
- <http://msdn.microsoft.com/msdnmag/issues/03/11/AuthorizationManager/default.aspx>
- <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/windowsserver2003/maintain/security/athmanwp.asp>
- http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/windowsserver2003/proddocs/entserver/authm_topnode.asp

