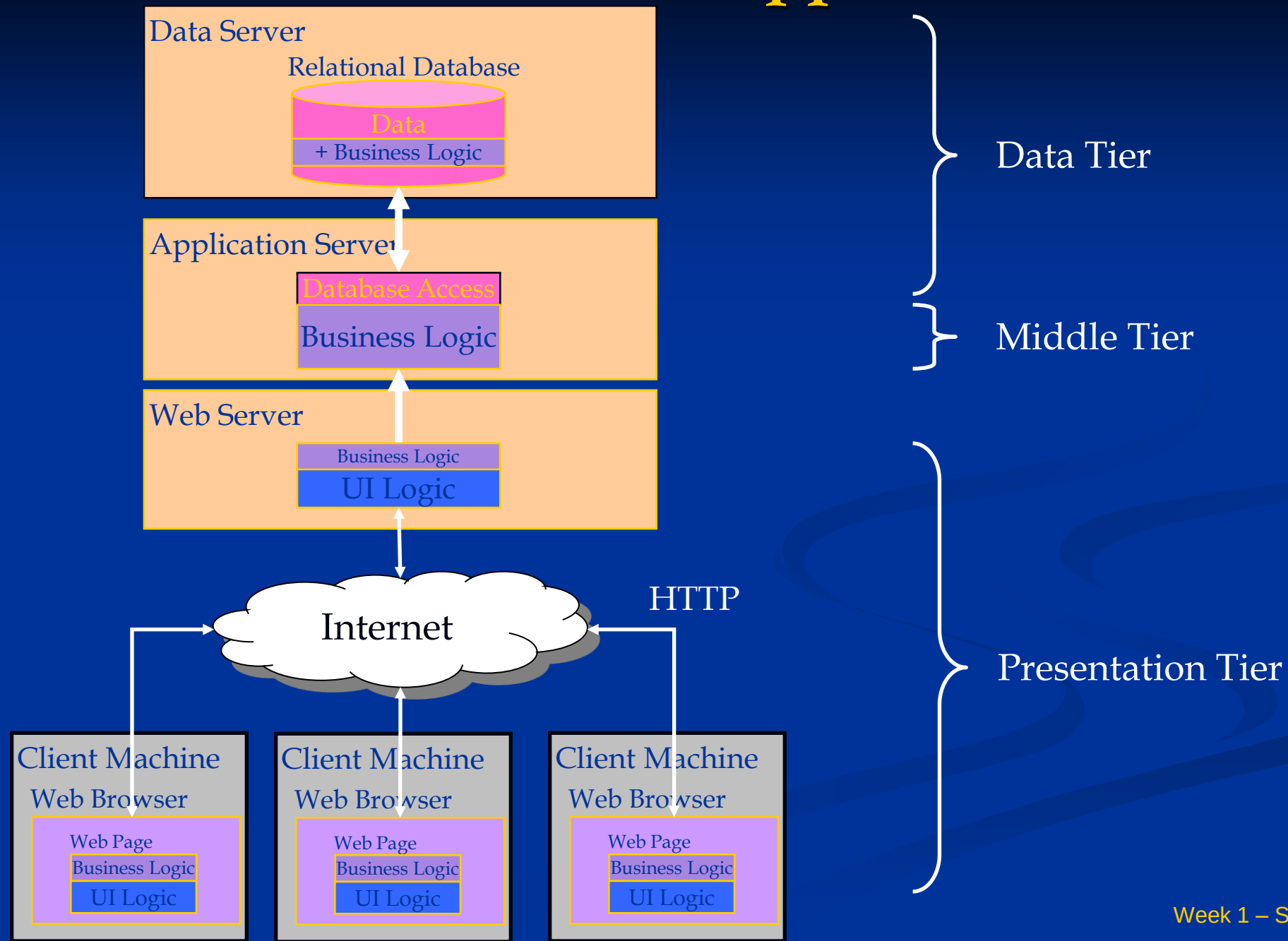
Web Security

Web-based Applications



Overview

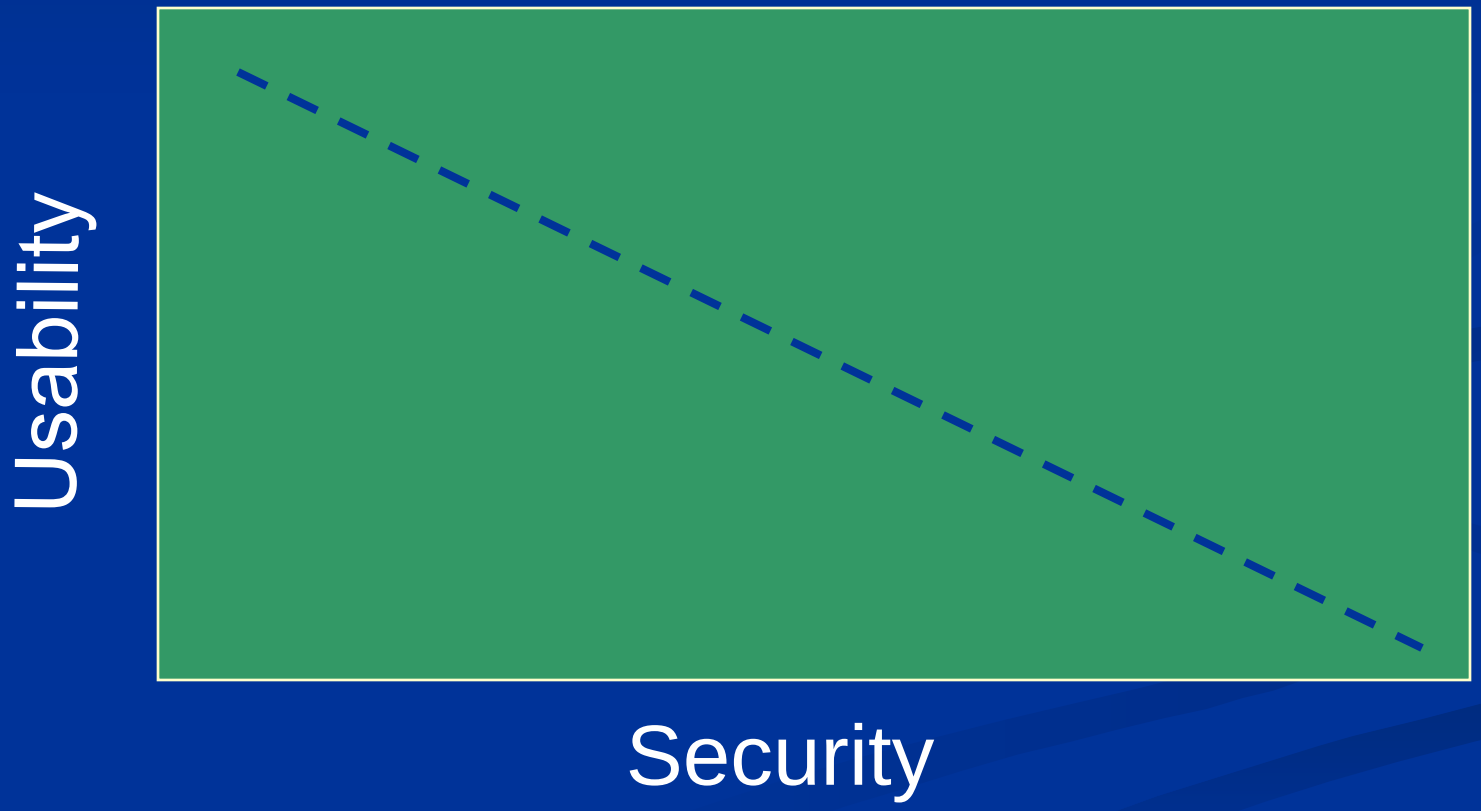
- Security Principles
- Windows Operating System Security
- Network Security
- Internet Explorer (IE) Security
- Internet Information Services (IIS) Security
- .NET Framework Security
- ASP.NET Security
- SQL Server Security

Why Is Security Difficult?

- Attacker need only find one weak point
- Defender needs to make sure that all possible entry points are defended
- Security is often an afterthought
- Usability of a system is inversely proportional to its security

Usability vs. Security

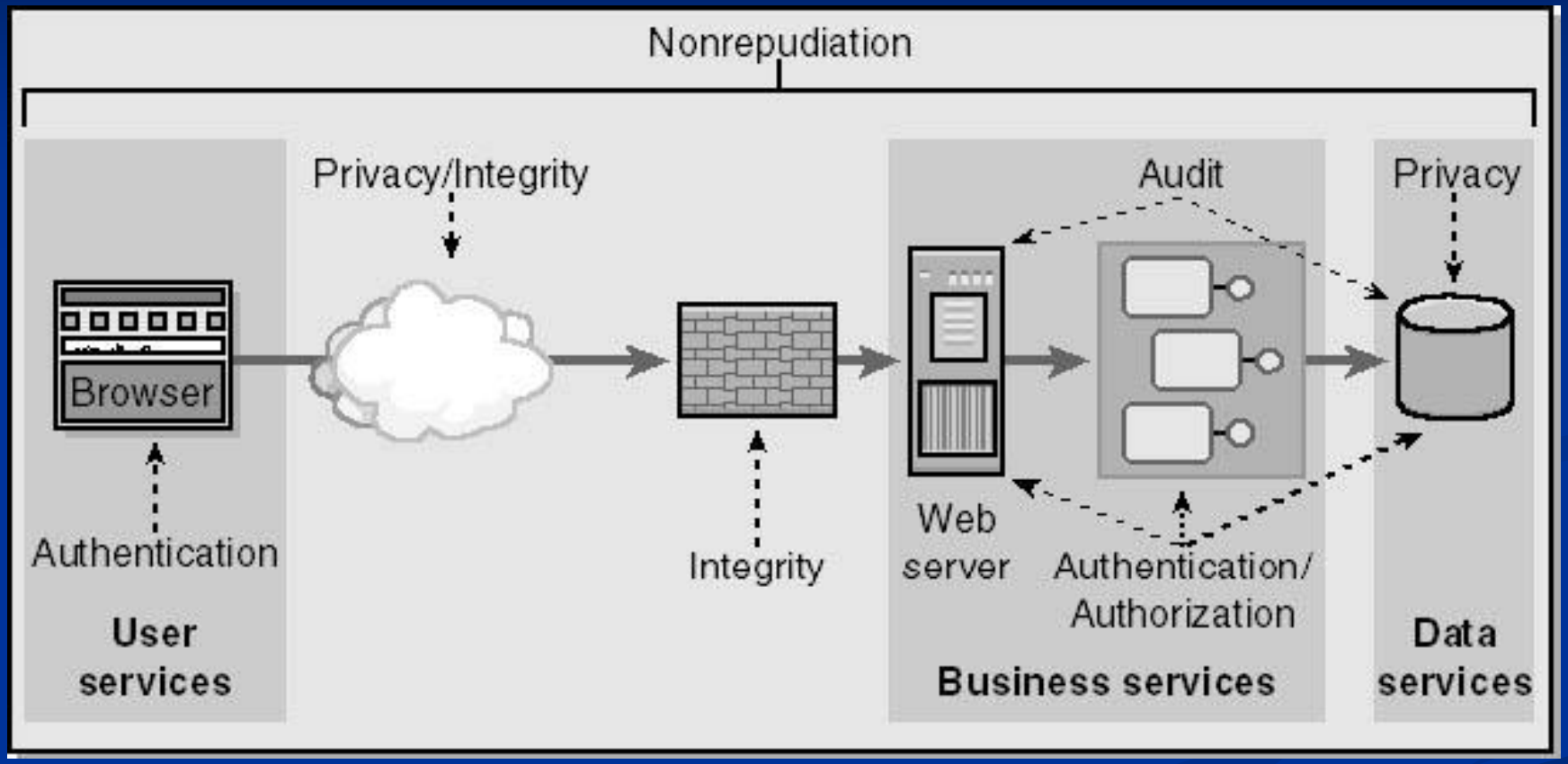
- The most secure computer system is turned off and buried in a concrete bunker!



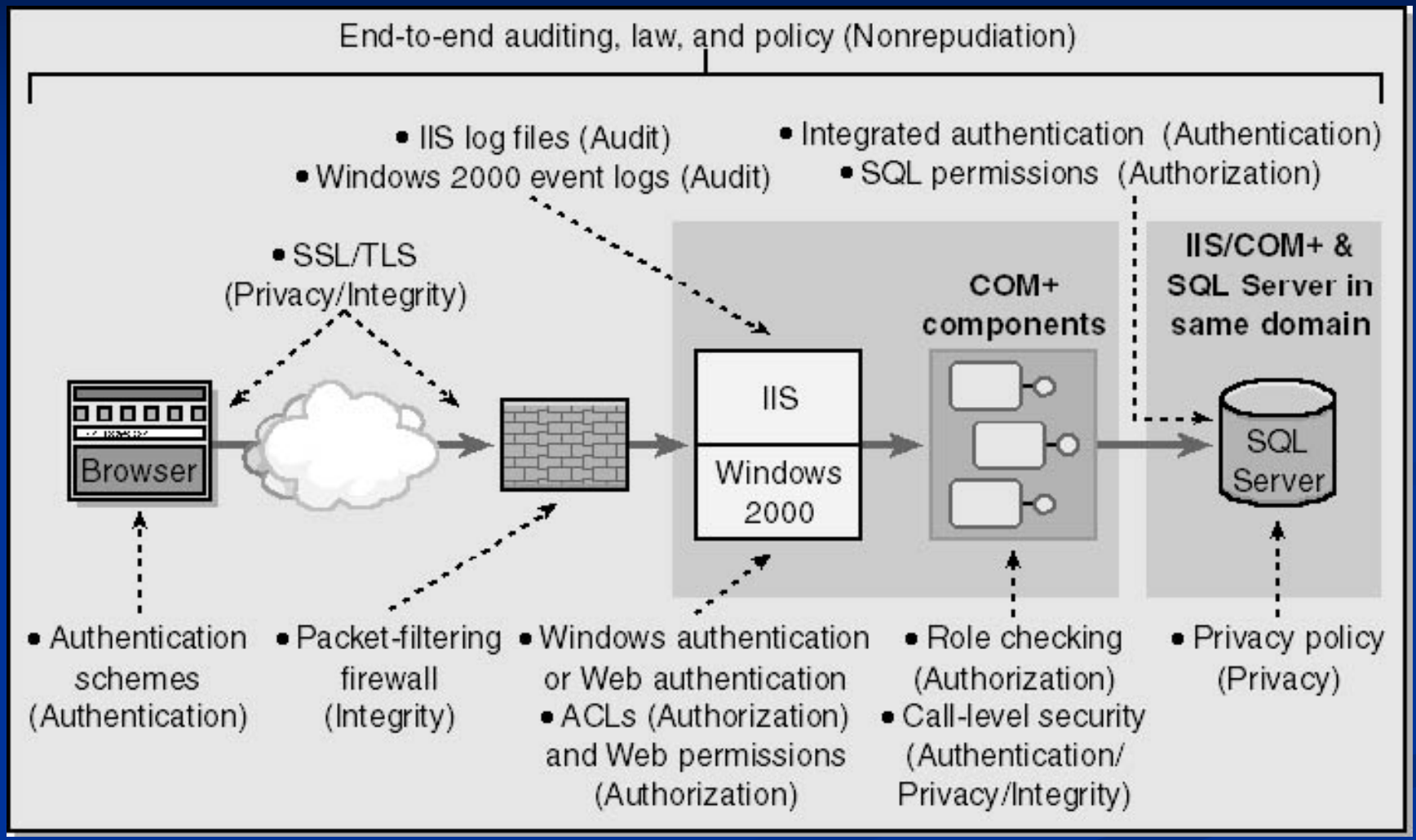
Security Terminology

- Authentication
  - principal tries to prove claimed identity (e.g. via password).
- Authorization
  - principal's identity then determines the permissions granted.
- Auditing
  - logging of successful and failed access attempts.
- Privacy
  - preventing unauthorized read access (e.g. encrypted network transfer)
- Integrity
  - detection of data tampering (especially in transit)
- Availability
  - preventing denial of service attacks
- Non-repudiation
  - providing binding proof if a transaction is disputed. E.g. digital signature

Web Application Security



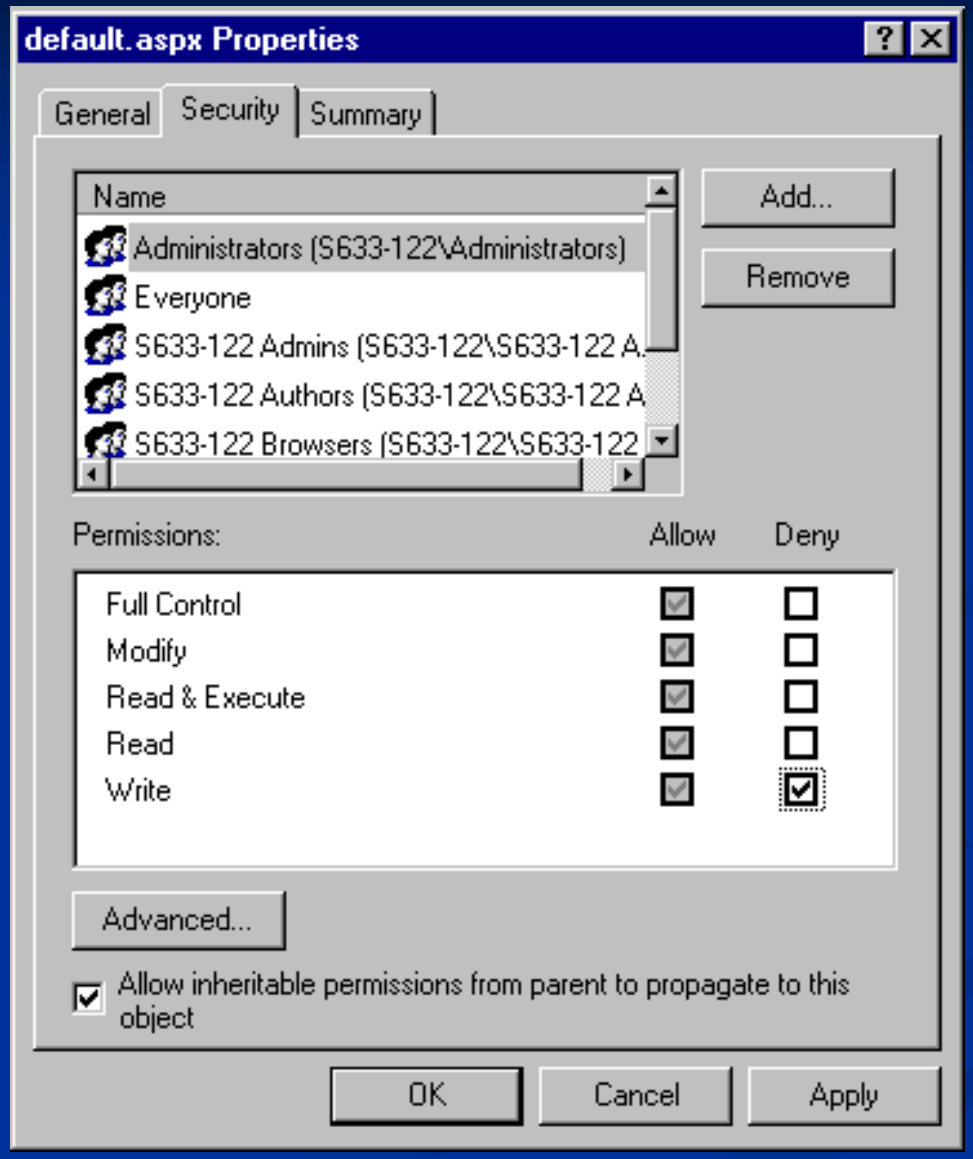
Web Application Security (Windows)



Windows Security Concepts

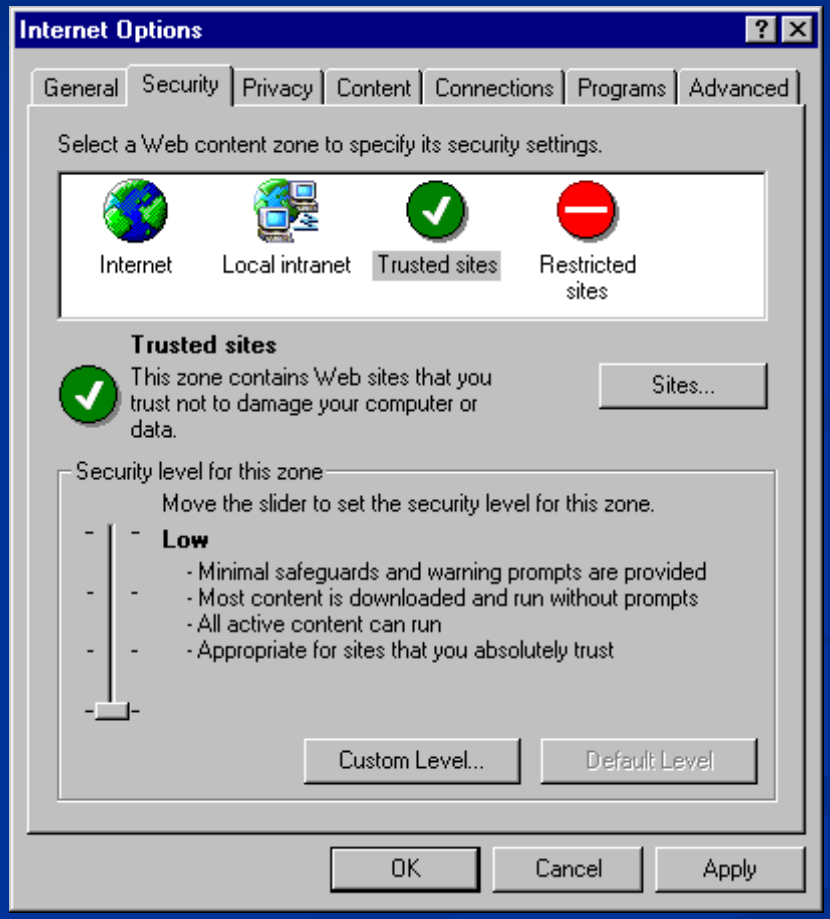
- Authentication
  - uses NTLM (Microsoft) or Kerberos (Active Directory) protocols
- User Accounts and Groups
  - users can belong to groups; permissions granted to users and groups.
- Domains
  - a collection of security principals (machines, users) with a centralized accounts database
- Access Control Lists
  - lists of "allow" or "deny" permission settings for each resource.
- Impersonation
  - allowing a thread to assume a different identity.

Access Control Lists



Internet Explorer Security

- Security Zones
  - URLs categorized into Zones; Zones assigned permissions



Reference:
http://windows.microsoft.com/en-au/windows-vista/change-internet-explorer-security-settings

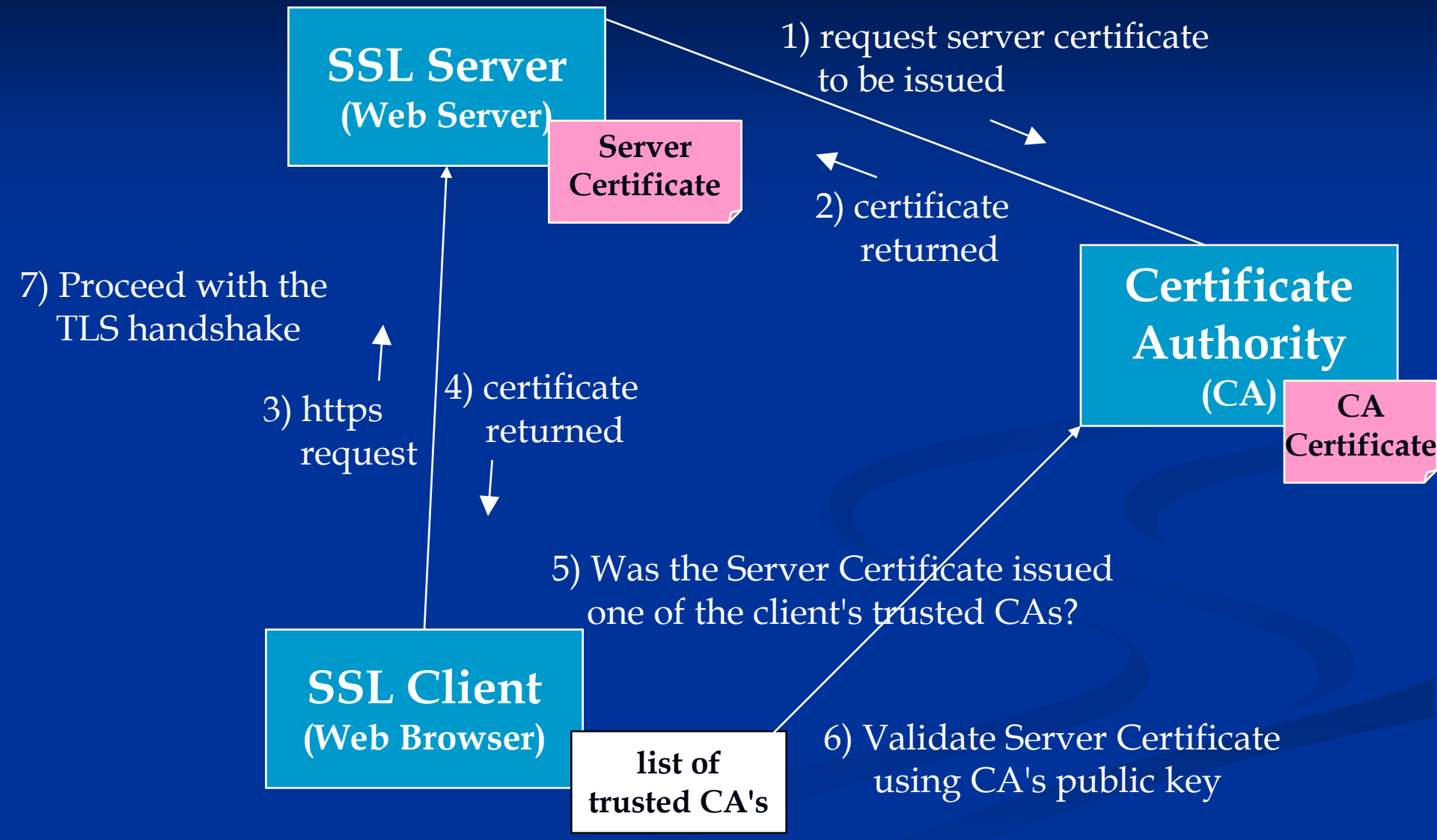
Transport Layer Security (TLS) – Ensuring Privacy and Integrity over un-trusted Networks

- Secure Sockets Layer (SSL)
  - a protocol developed by Netscape for transmitting private documents via the Internet.
  - designed to establish a secure connection between two computers
  - by convention, URLs that require an SSL connection start with `https:`
- Standardized as ***Transport Layer Security Protocol (TLS)*** by the ***Internet Engineering Task Force (IETF)***
  - http://tools.ietf.org/html/rfc5246 (updated 2008)
  - http://tools.ietf.org/html/rfc6176 (updated 2011)

Transport Layer Security (TLS) Properties

- The peer's identity can be authenticated
  - uses public key cryptography together with trusted authorities.
- The connection is private.
  - Encryption is used after an initial handshake to agree on a secret key.
- Message integrity is ensured.
  - a secure hash function is used to detect data tampering.

TLS Server Authentication

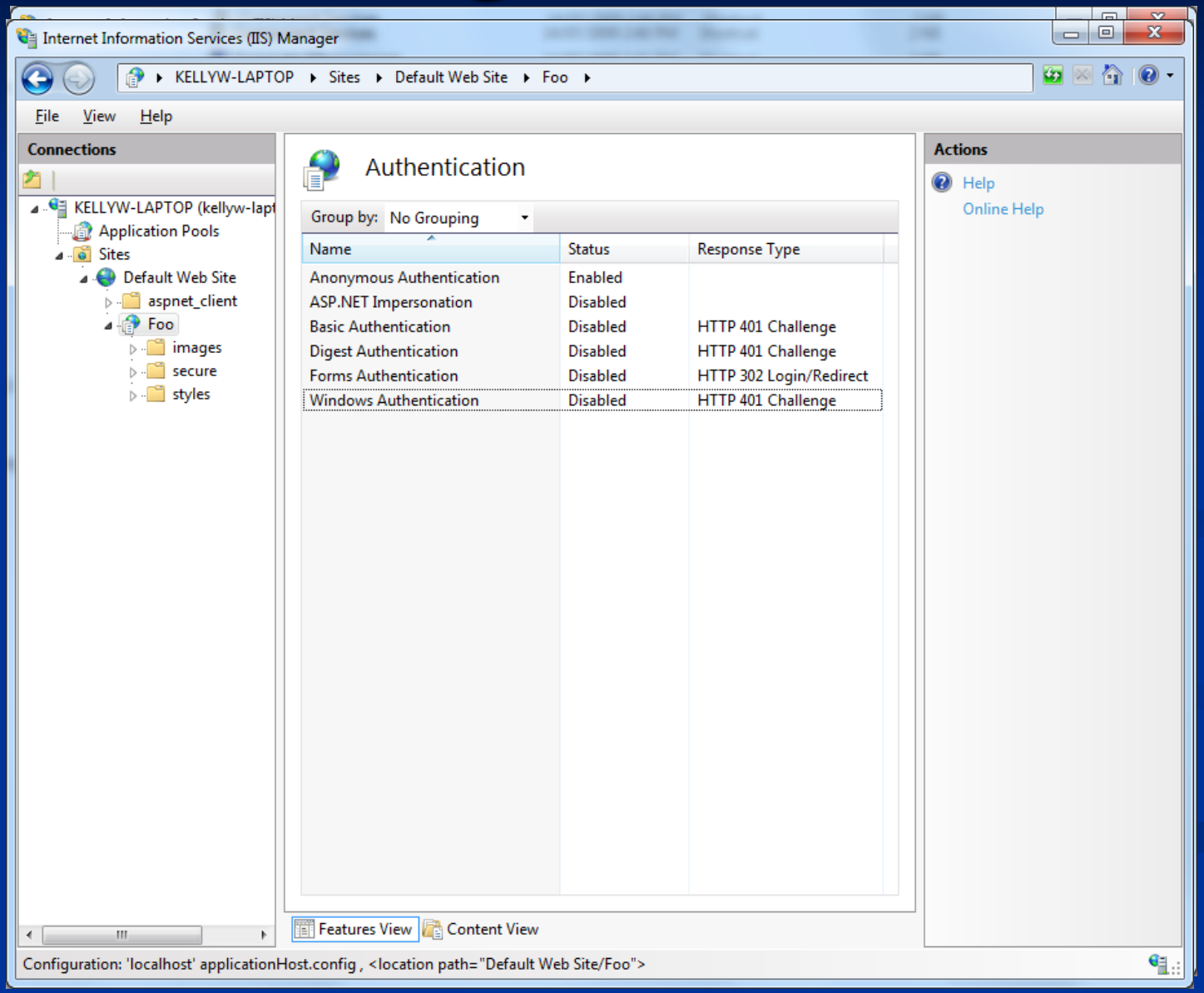


IIS Security

IIS supports the following authentication protocols:

- Anonymous
  - no authentication
- Basic (part of the HTTP 1.0 specification)
  - clear text username and password (insecure).
- Digest
  - hashed username and password
- Integrated Windows
  - NTLM or Kerberos
- SSL client certificates
  - uses X.509 digital certificate standard
- IIS 7 new: Form authentication (Login-redirection)

IIS Management Console

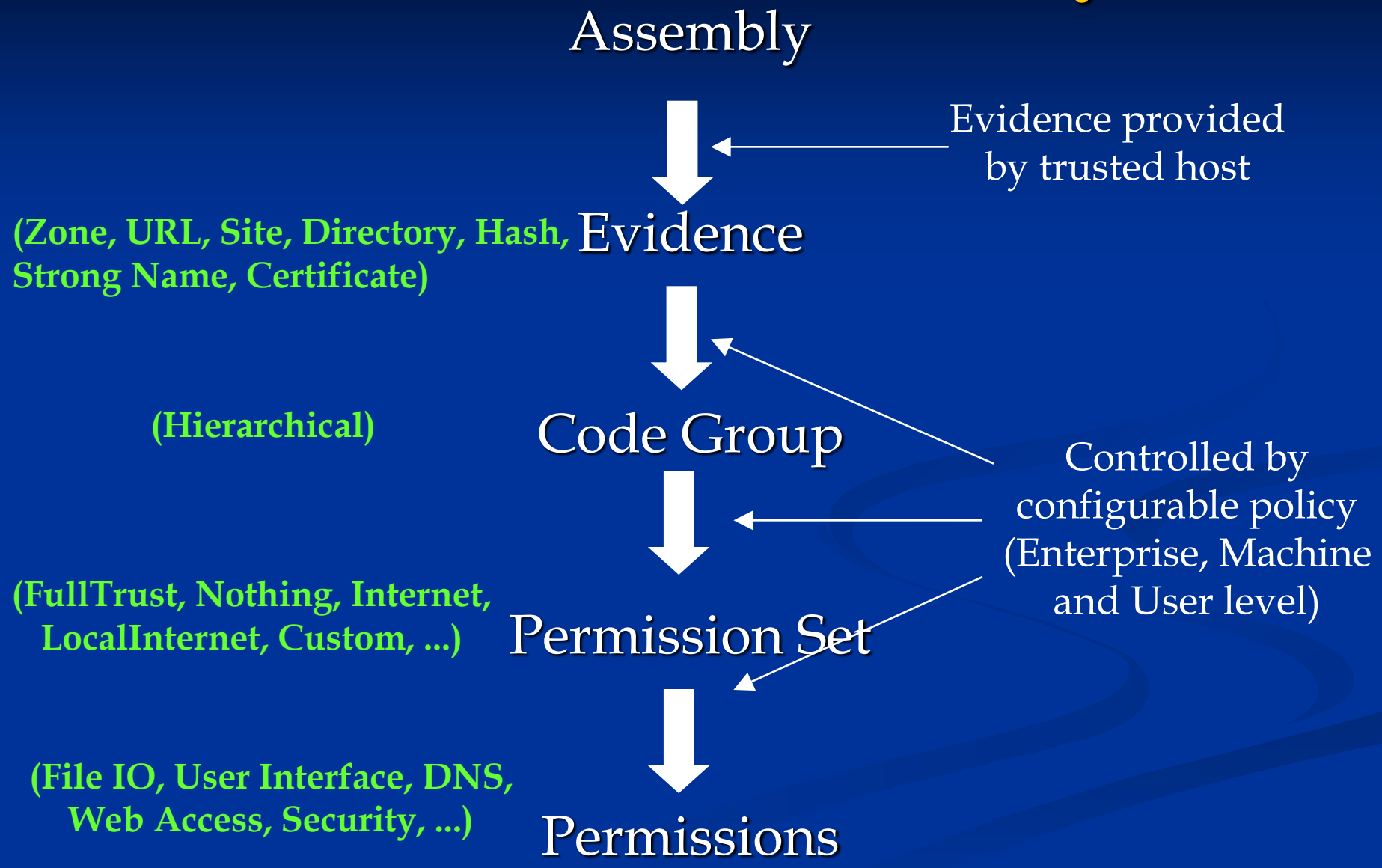


.NET Framework Security

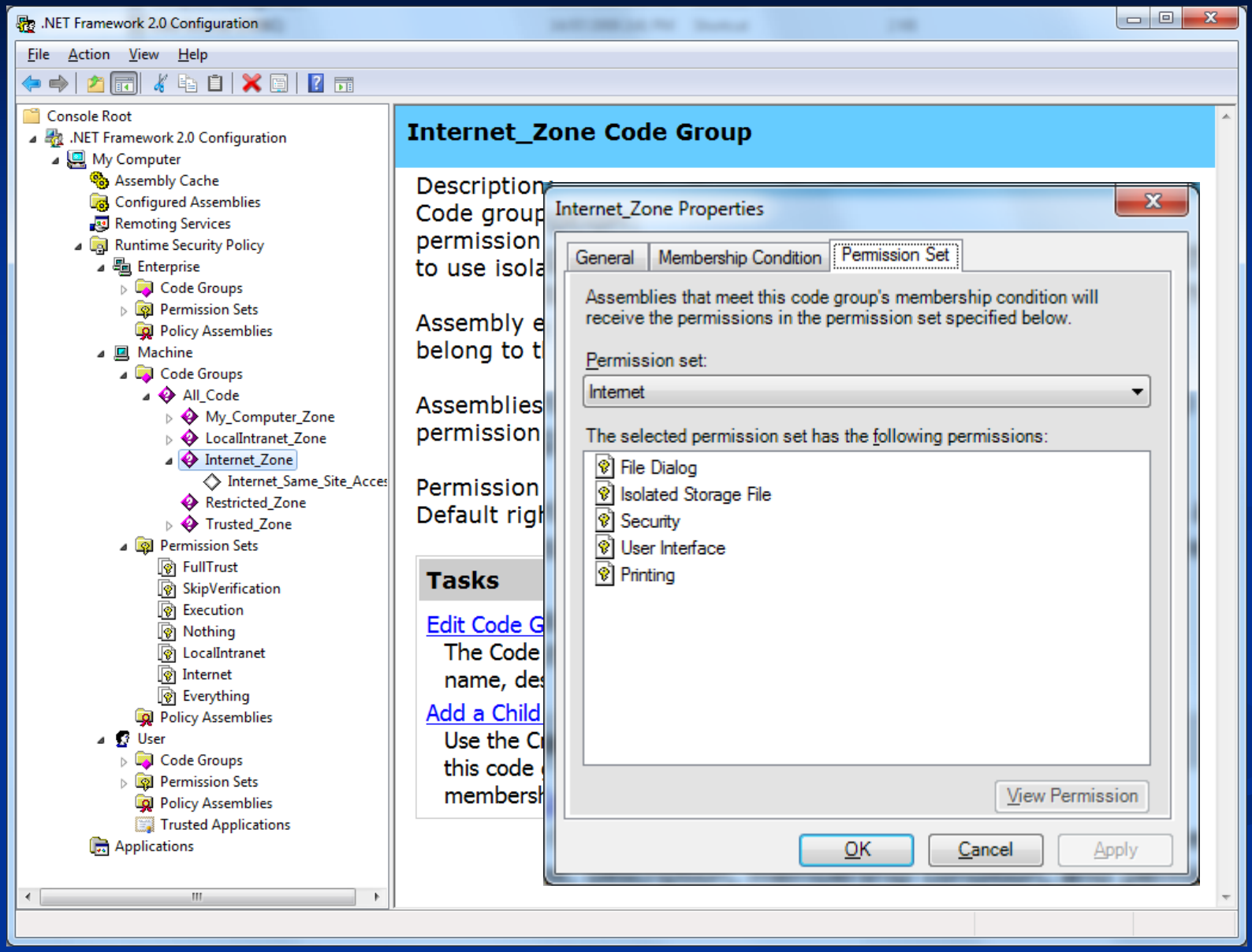
- Extra security layer on top of operating system security.
- Depends on verifiable managed code so that the common language runtime (CLR) can enforce security policy.

- Role-based security:
  - Permissions granted based on the role of the user (principal) executing the code.
- Code access security:
  - Permissions granted based on how well the piece of code (assembly) is trusted.

Code Access Security



Runtime Security Policy Management Console



Demanding Permissions

- Trusted system libraries demand that all callers have appropriate permissions in order to use them.
  - Eg the `System.IO.FileStream` class demands `FileIOPermissions`
- Normal user code can also demand that callers have arbitrary sets of permissions.

Declarative vs Imperative Security

- Declarative Security:
  - Attributes applied at the assembly, class or method level
  - e.g.

```
[assembly:FileIOPermissionAttribute (SecurityAction.RequestMinimum,
  Write="C:\\test.tmp")]
```

- Imperative Security:
  - Can be executed conditionally.
  - e.g.

```
FileIOPermission p =
     new FileIOPermission(FileIOPermissionAccess.Write,
                          "C:\\test.tmp");
p.Demand();
```

Role Based Security

- The principal:
  - has a unique identity (name and authentication type)
  - plays zero or more roles.
- The authentication of a principal's identity can be:
  - based on a Windows account (WindowsIdentity)

```
string name = WindowsIdentity.GetCurrent().Name;
```

  - or custom (GenericIdentity)

```
GenericIdentity MyIdentity = new GenericIdentity("MyIdentity");
string[] MyRoles = new string[] {"Manager", "Teller"};
Thread.CurrentPrincipal = new GenericPrincipal(MyIdentity, MyRoles);
```

- Security Checks can be performed directly on the Principal:

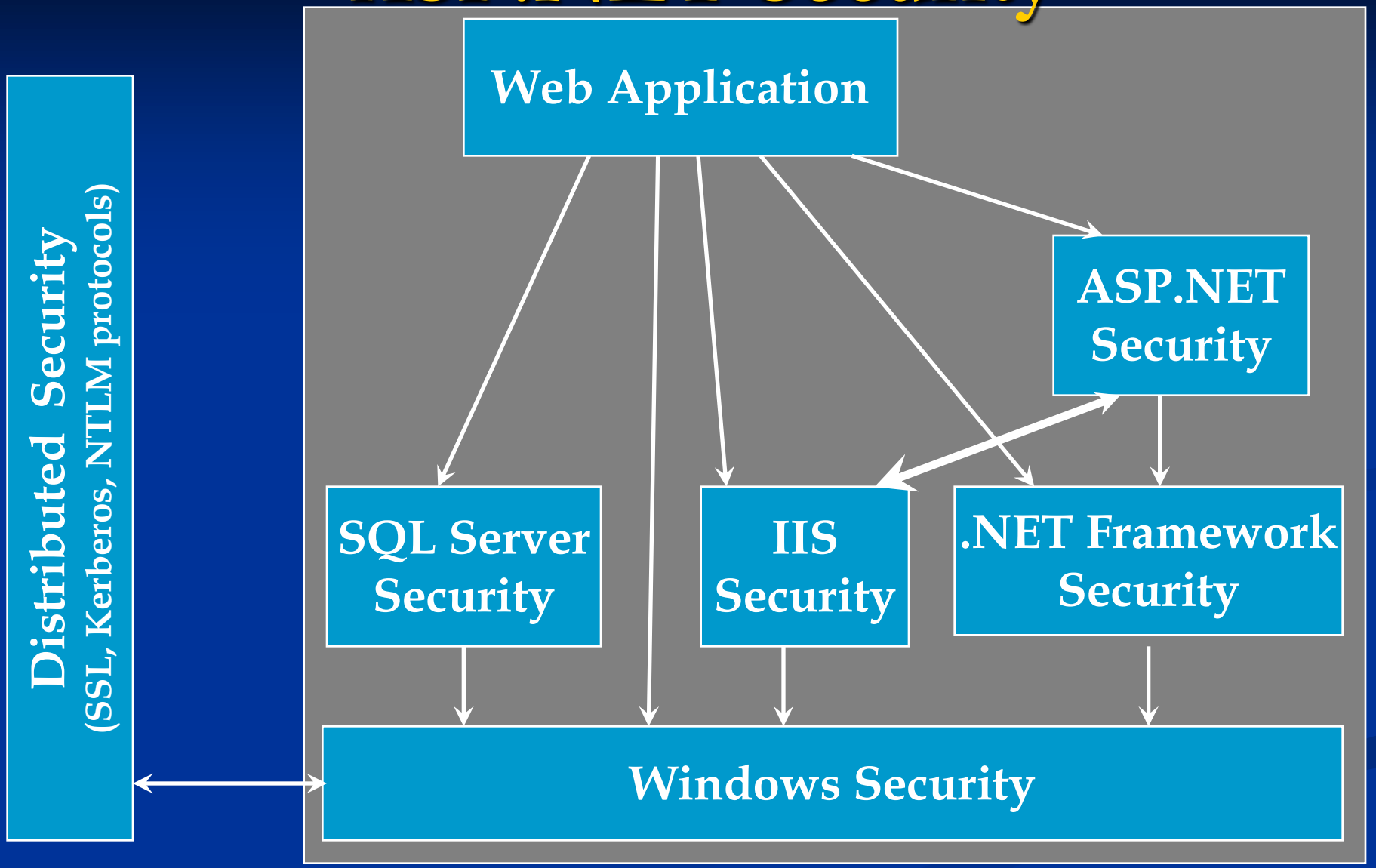
```
if (Thread.CurrentPrincipal.IsInRole("Manager"))
```

- Or the PrincipalPermission class can be used to do code access security style checks (either imperatively or declaratively):

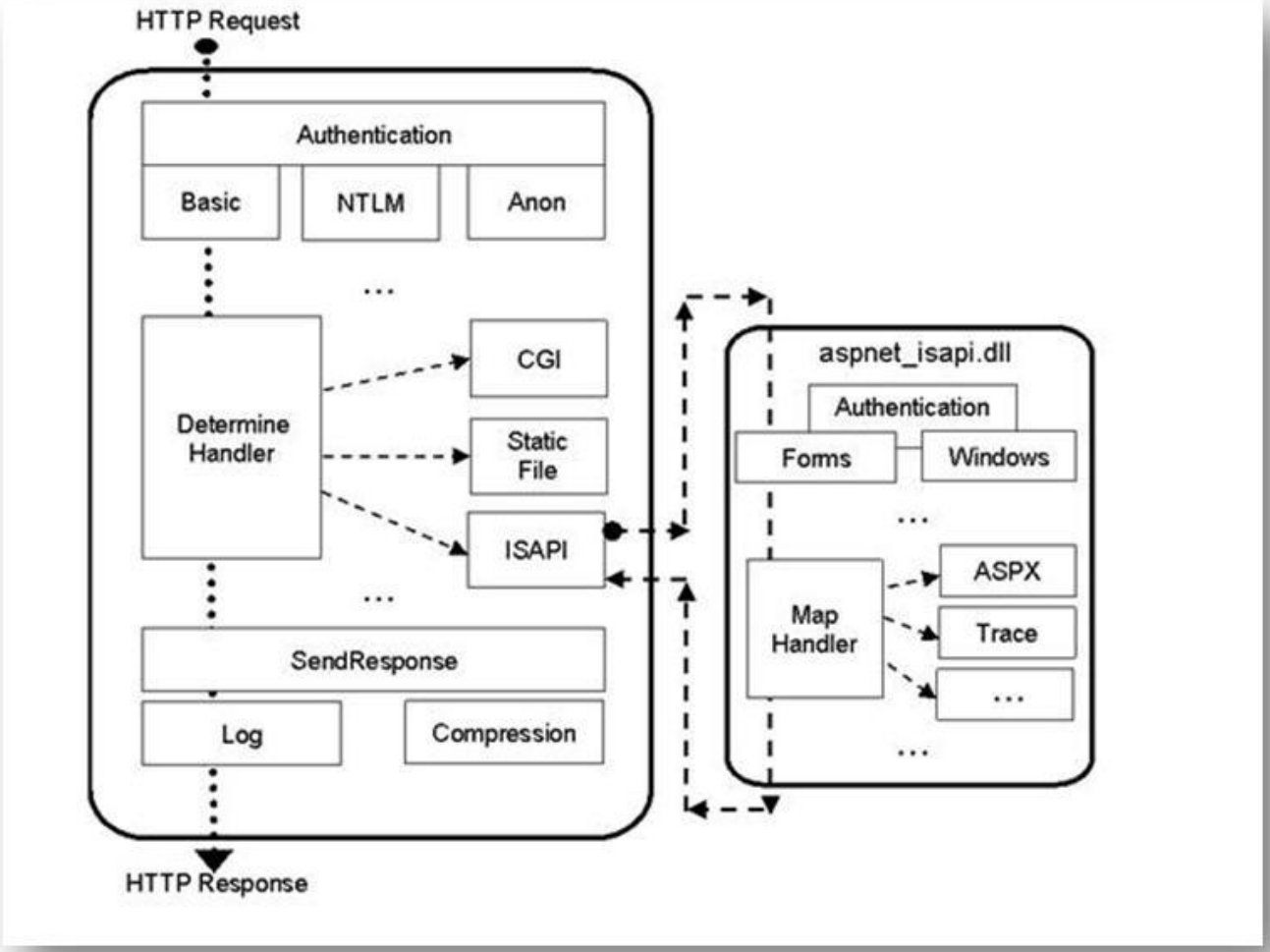
```
(new PrincipalPermision(null, "Manager")).Demand();

[PrincipalPermission(SecurityAction.Demand, Role="Manager"]
```

ASP.NET Security

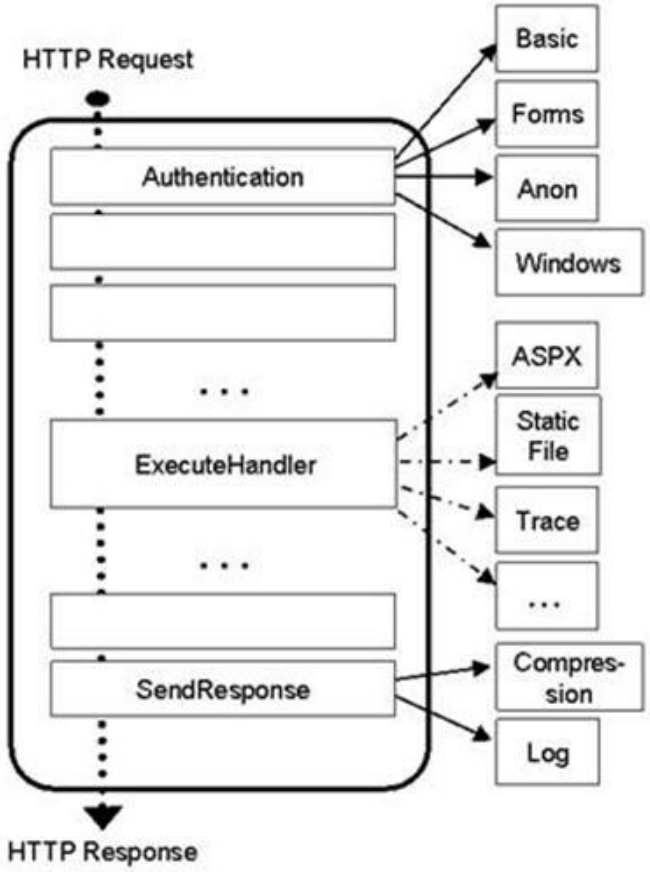


Classic IIS 6 Process Model

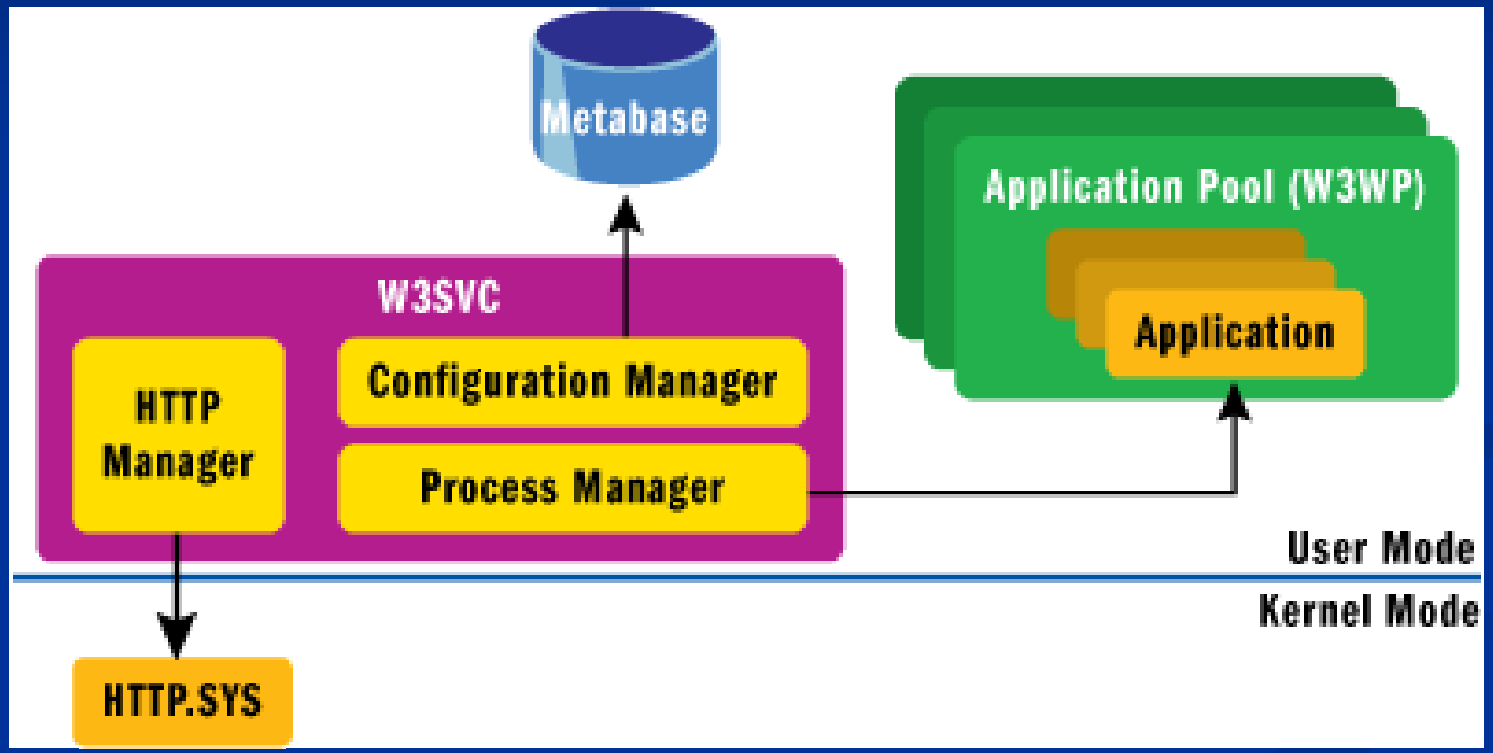


http:///learn.iis.net/page.aspx/243/aspnet-integration-with-iis-7

IIS 7 Integrated Pipeline

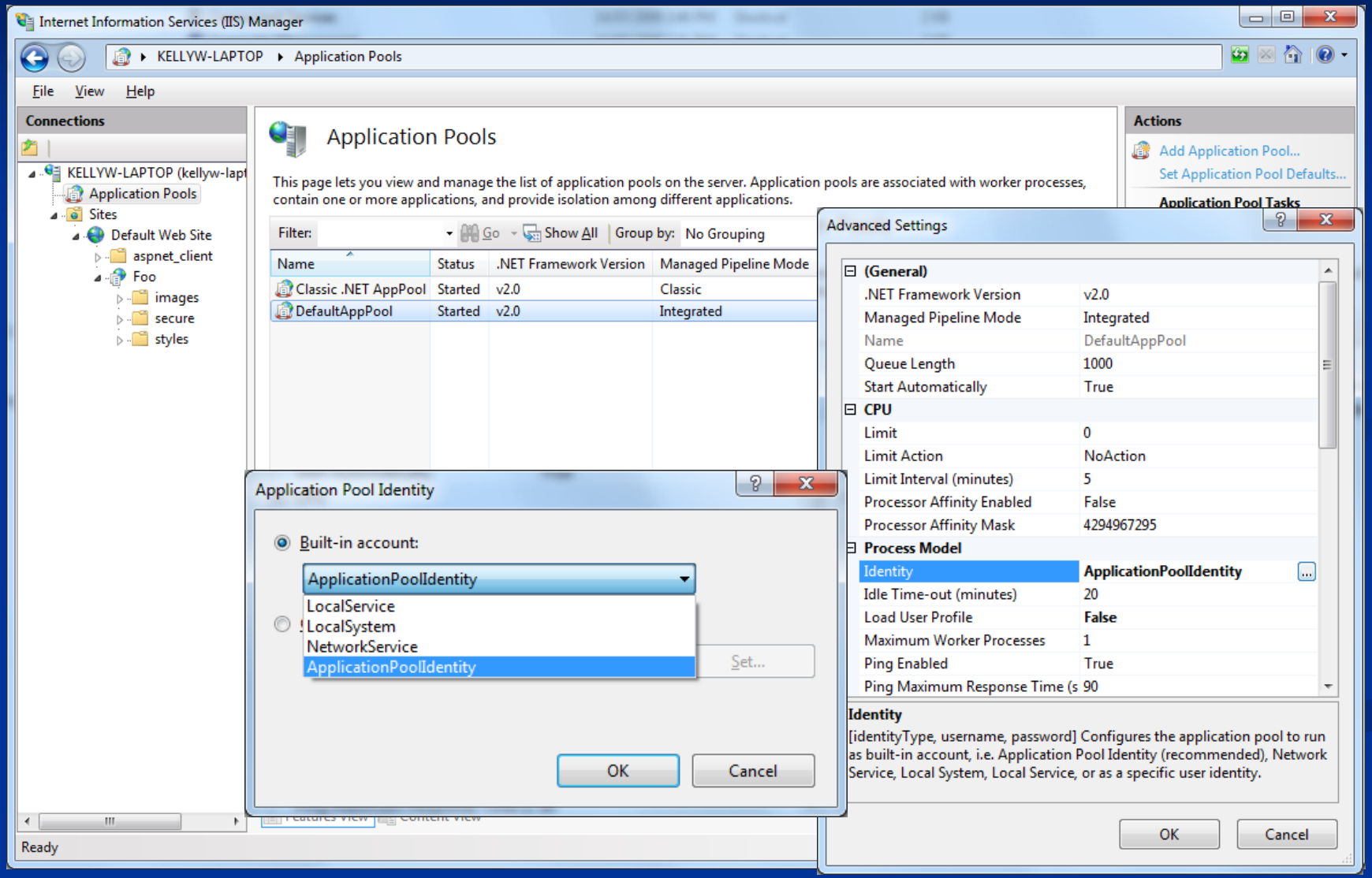


IIS 7 Worker Processes

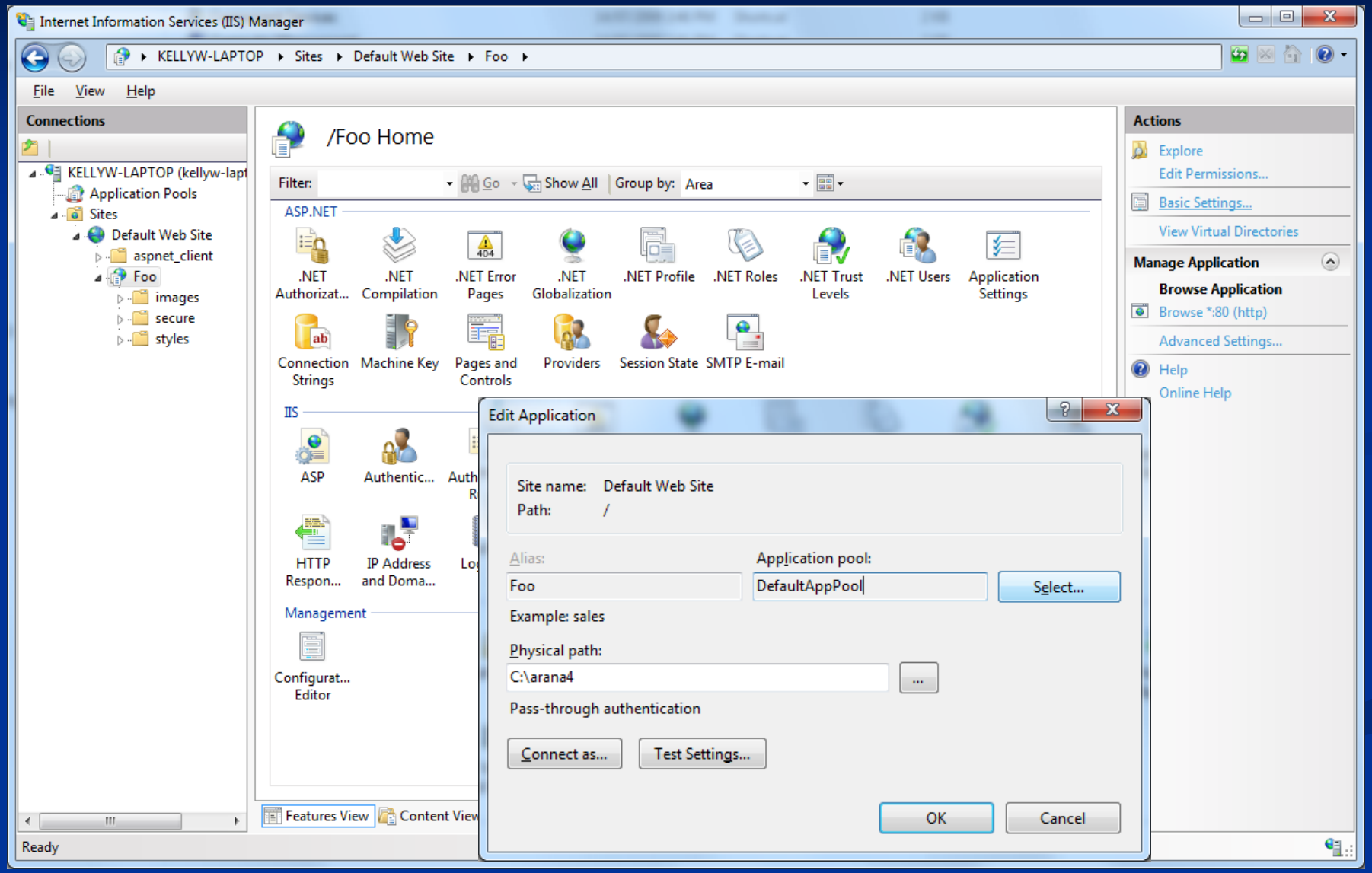


http://msdn.microsoft.com/en-us/magazine/cc163357.aspx

Application Pools



Application Pools



ASP.NET Authentication Choices

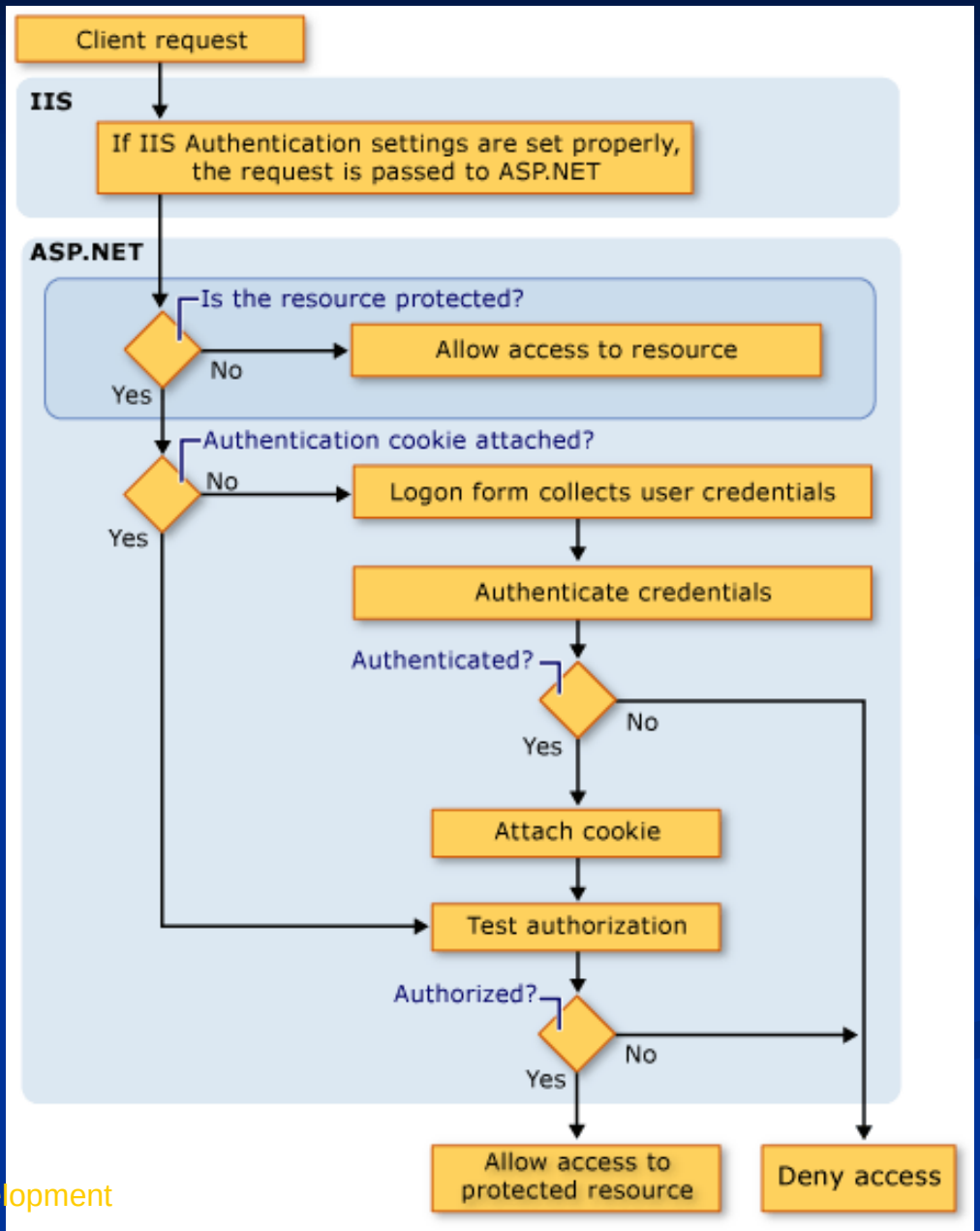
1. **None – Default Authentication (Anonymous access)**
2. **ASP.NET Windows Authentication**
   - Requires all users to have an account on the Windows Domain.
   - Only suitable for intranets.
   - Relies on IIS web server doing the Authentication
   - Eg: Access to TeamWorker
3. **ASP.NET Forms Authentication**
   - Allows non-Windows users to authenticate
   - Normally configure IIS to allow anonymous access.
   - Uses a login web page for user to enter credentials
   - When authenticated users are given an encrypted cookie to identify themselves on subsequent requests to the web site.
   - Have the option of allowing users to self register for an account.
4. **Passport Authentication – centralised service provided by Microsfot, single sign on capability across multiple domains**

Configuring ASP.NET Forms Authorization

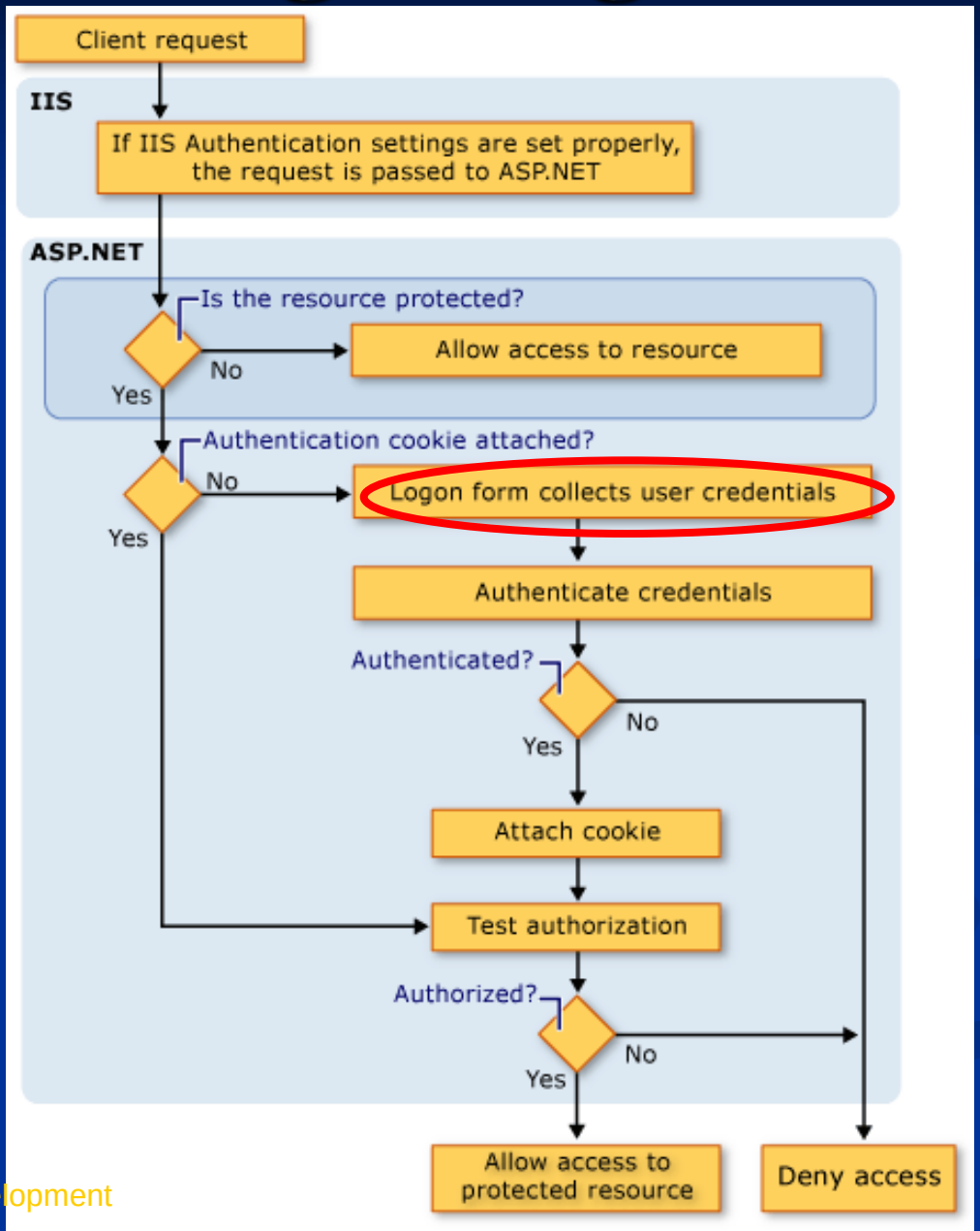
Web.config* | Default.aspx

```
<?xml version="1.0"?>
<configuration>
    <system.web>
      <authentication mode="Forms">
        <forms loginUrl="Login.aspx"/>
      </authentication>
    </system.web>
</configuration>
```

ASP.NET Forms Authentication

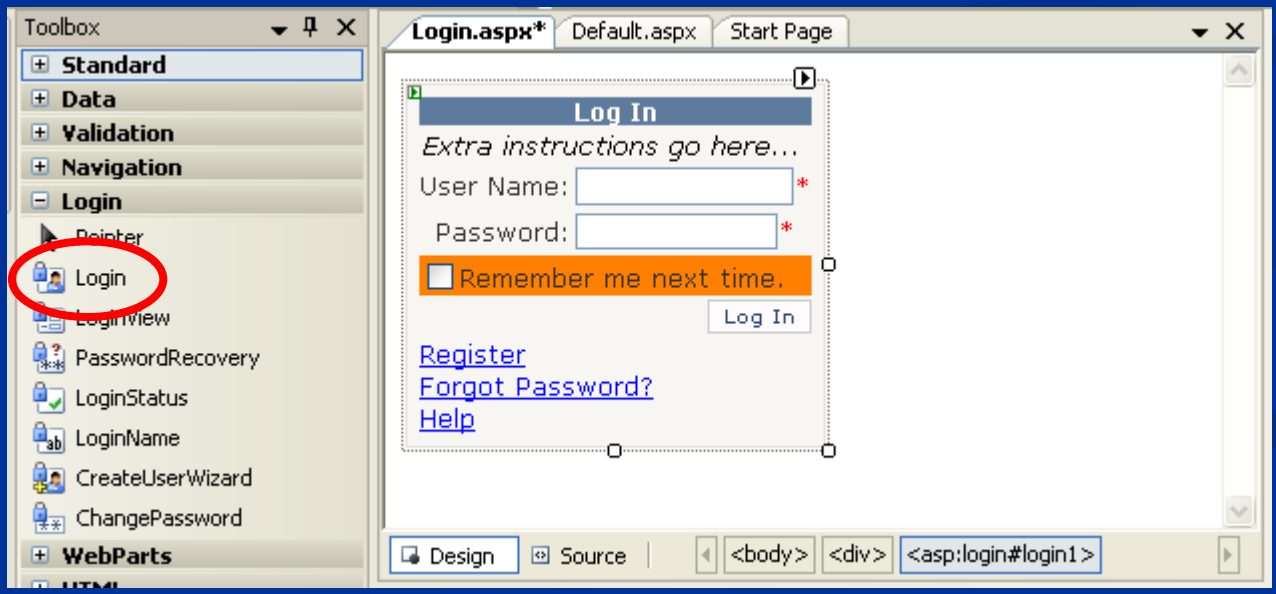


Login Page

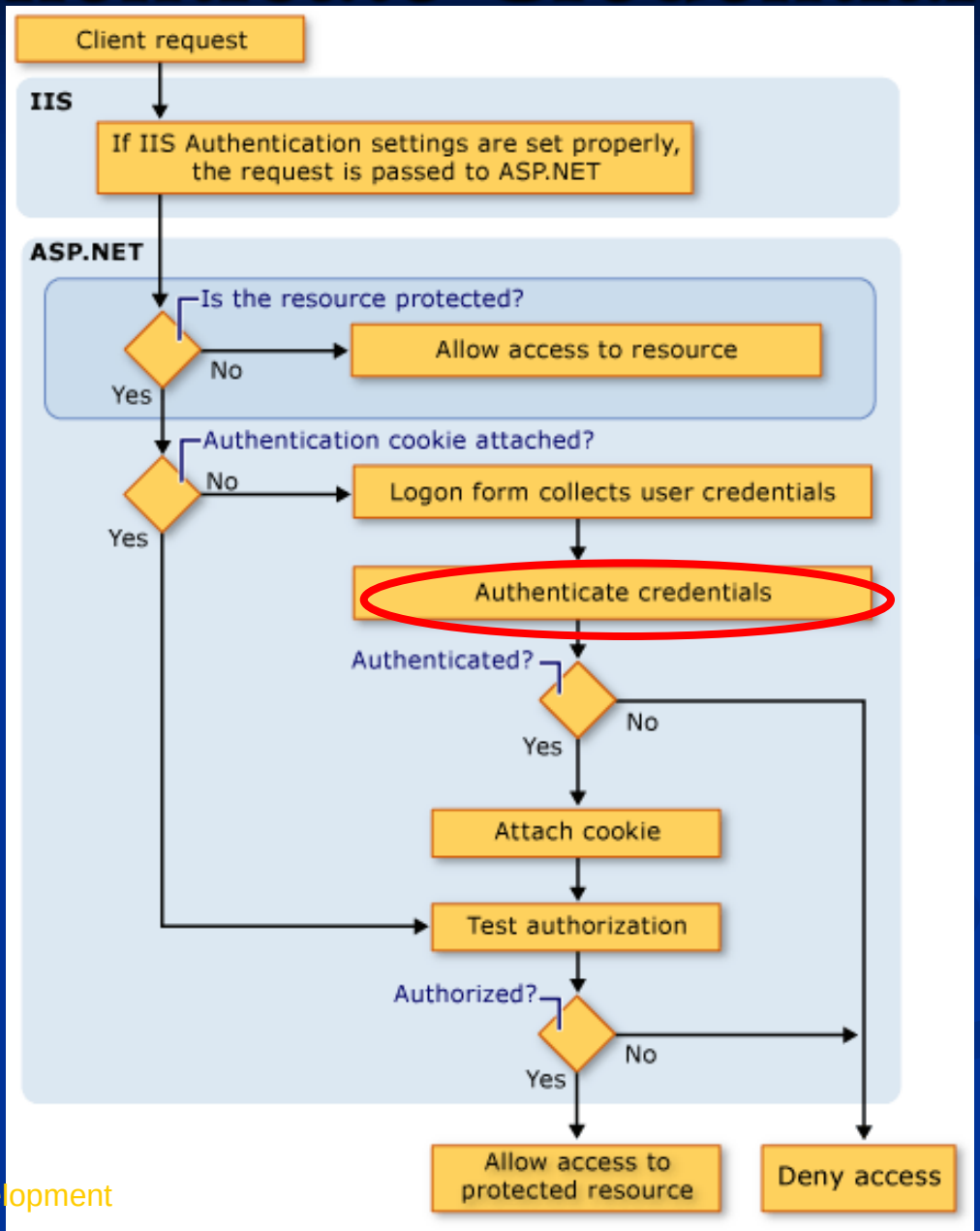

Client request
IIS
If IIS Authentication settings are set properly, the request is passed to ASP.NET
ASP.NET
Is the resource protected?
No
Allow access to resource
Yes
Authentication cookie attached?
No
Logon form collects user credentials
Yes
Authenticate credentials
Authenticated?
No
Yes
Attach cookie
Test authorization
Authorized?
No
Yes
Allow access to protected resource
Deny access

Login Page Options

1. Create a Login page manually:
   - Typically provide textboxes for user name and password.
   - Manually write code to authenticate user and issue cookie
2. Create a page containing an **ASP.NET Login control**:
   - Appearance and behaviour can be configured
   - Can avoid writing any code.



Authenticate Credentials



Authentication Options

1. Write your own custom authentication code
   - Typically accesses a database to check if password matches user name.
   - Can be done by business tier in N-tier architecture.
   - Passwords should be stored encrypted.
2. Use **ASP.NET Membership services**
   - Can automatically create a SQL Server database and tables for storing user name, encrypted password, etc.
   - Automatically accesses this database to perform authentication.

ASP.NET Forms Authentication & Authorization

Uses cookies to keep track of authenticated users, redirects to login page if authentication required

ASP.NET Forms Authentication & Authorization

Auto generated

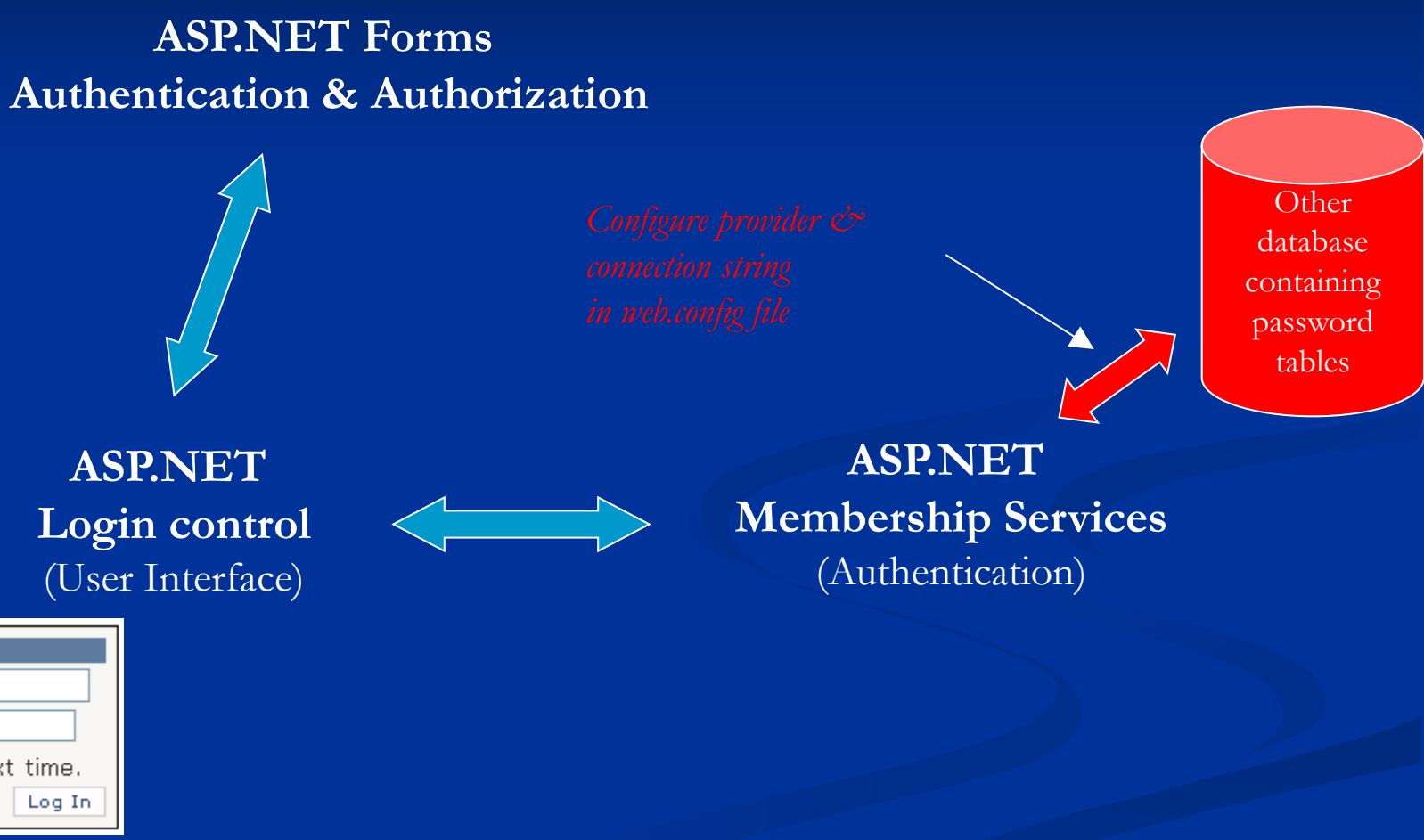
ASP.NET Login control
(User Interface)

ASP.NET Membership Services
(Authentication)

Checks if password is correct for user

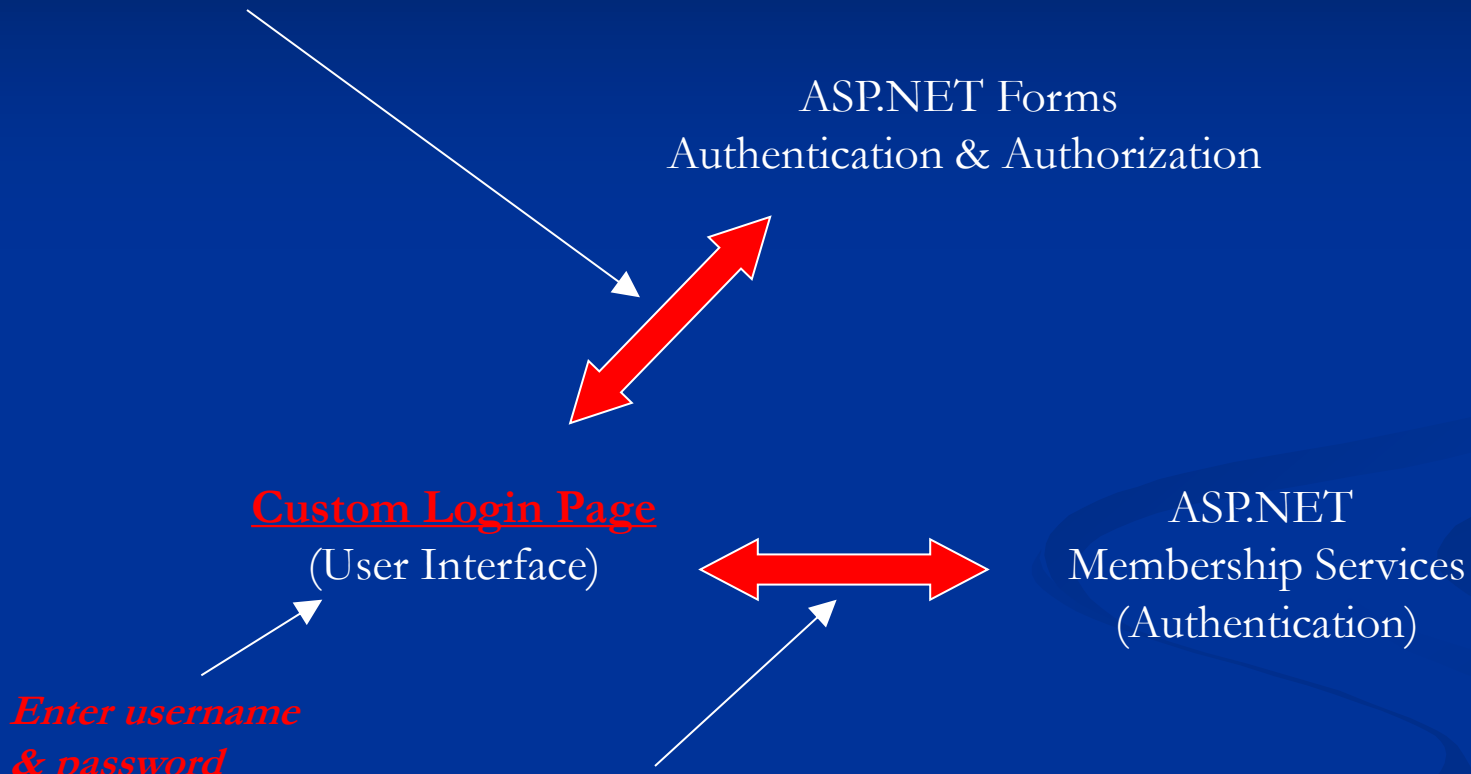
Database containing passwords

ASP.NET Forms Authentication & Authorization



ASP.NET Forms Authentication & Authorization

```
FormsAuthentication.RedirectFromLoginPage(username, persist);
```




```
if (Membership.ValidateUser(username, password))
```

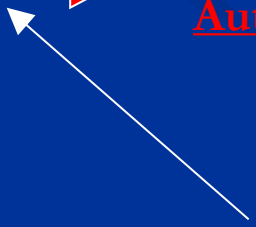
ASP.NET Forms Authentication & Authorization

ASP.NET Forms
Authentication & Authorization

ASP.NET
Login controls
(User Interface)

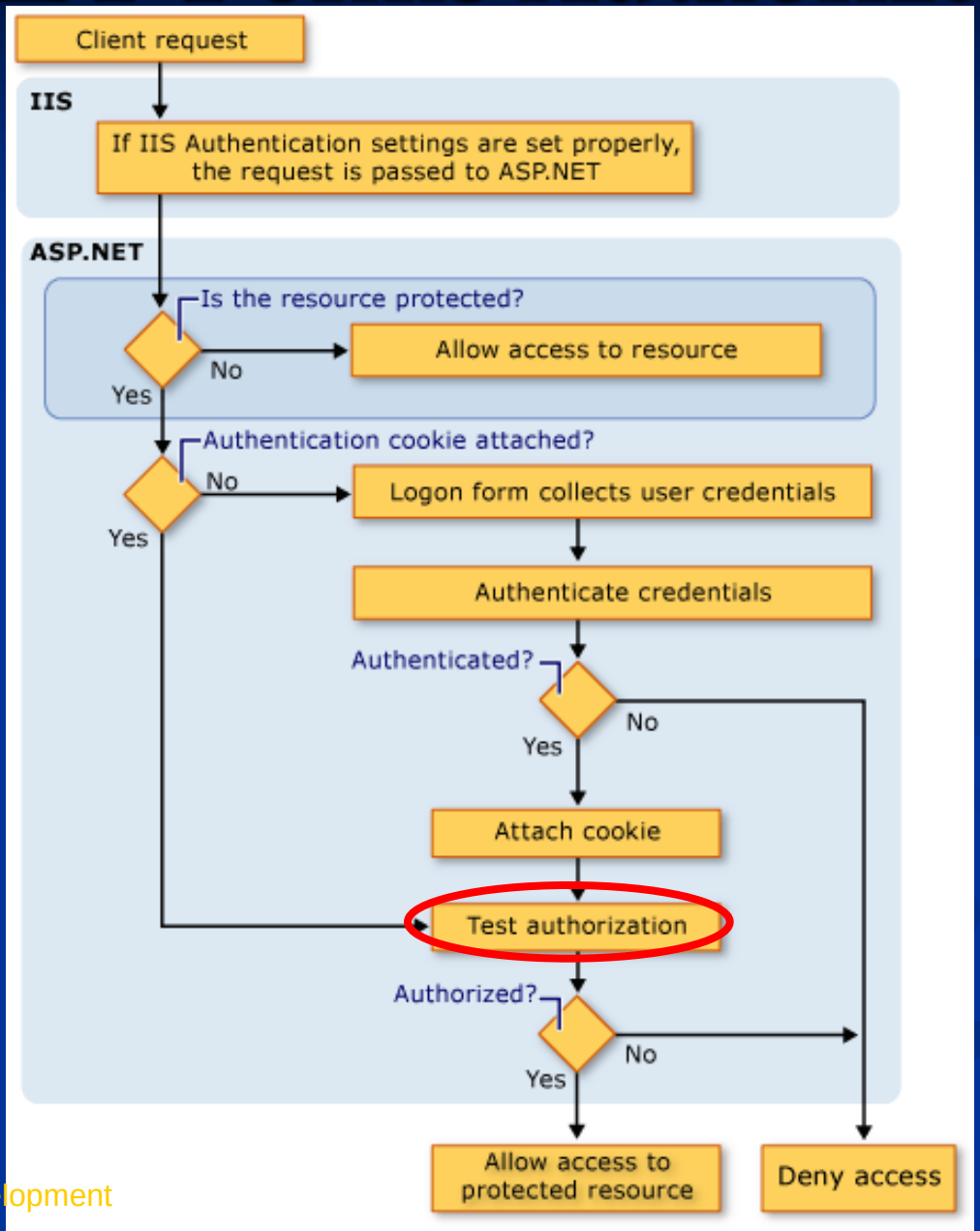
Custom
Authentication

Business
Tier

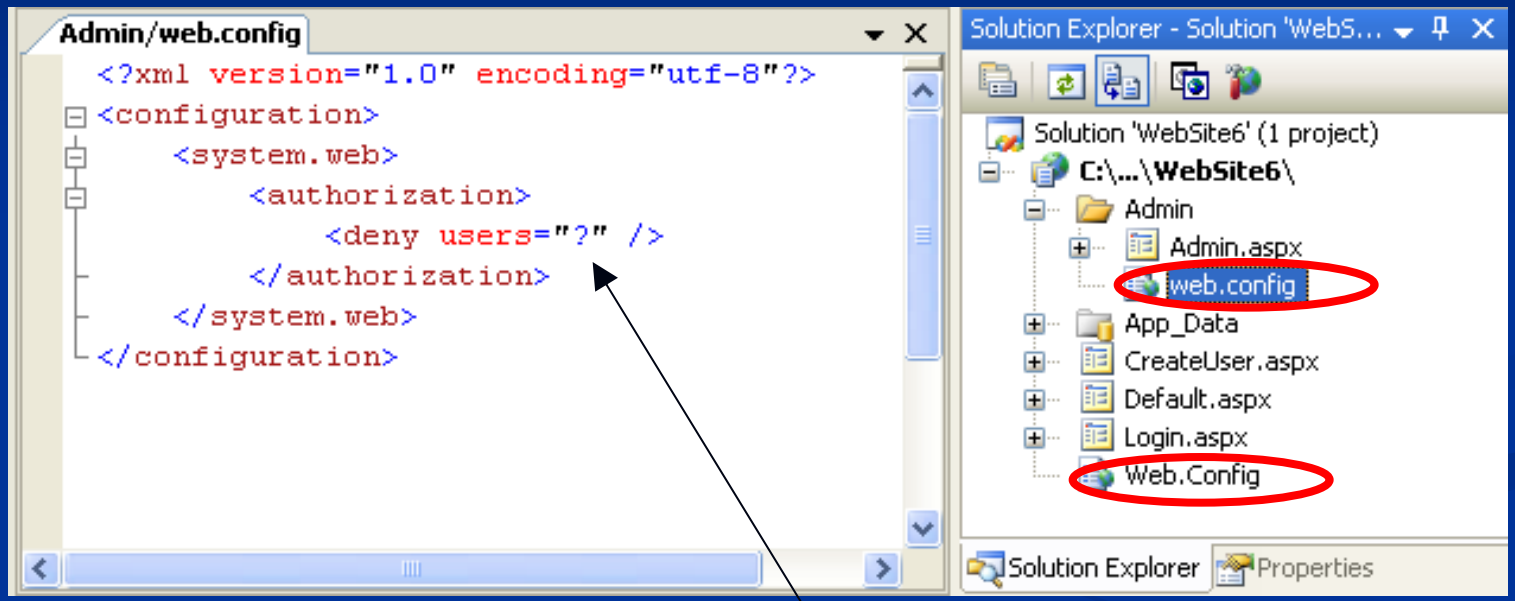


Implement handler for **`Authenticate`** *event*

ASP.NET Forms Authorization



Configuring ASP.NET Forms Authorization (Web.Config)




```
? = Anonymous
* = All users
```

User Identity

- Once the user has successfully authenticated, you can access the identity of the currently connected user from any (code behind) page within the web site:
  - `User.Identity.Name`
  - `User.Identity.AuthenticationType`
  - `User.Identity.IsAuthenticated`
  - `User.IsInRole("Administrator")`

Role Based Authorization

Admin/web.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <system.web>
        <authorization>
            <allow roles="Administrator" />
            <deny users="*" />
        </authorization>
    </system.web>
</configuration>
```

Order is important:
first rule takes priority!

Role Providers

- The roles of each user can be managed automatically by ASP.NET membership services.
- If you're not using ASP.NET membership services and you wish to support Role based security, you'll have to implement your own Role Provider:

Web.config

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <roleManager enabled="true" defaultProvider="MyRoleProvider">
      <providers>
        <add name="MyRoleProvider" type="MyRoleProvider,Business"/>
      </providers>
    </roleManager>
```

MyRoleProvider.cs*

MyRoleProvider | GetRolesForUser(string username)

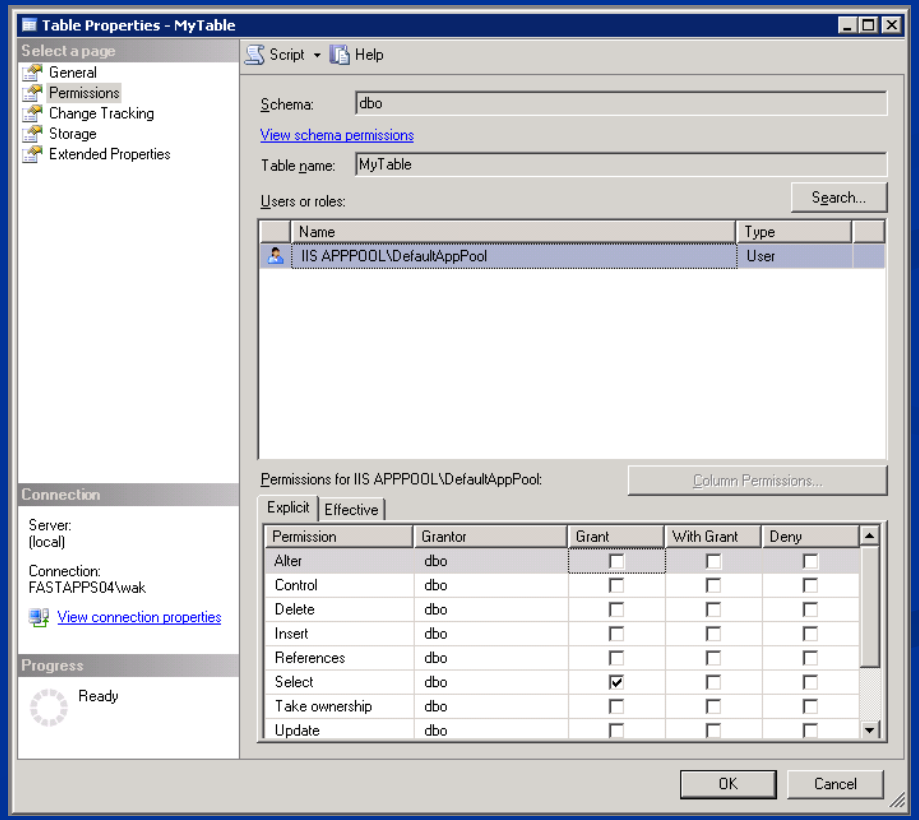
```
using System.Web.Security;

public class MyRoleProvider: RoleProvider
{
    public override string[] GetRolesForUser(string username)
    {
        if (username == "Admin")
            return new string[] { "Administrator" };
        else
            return new string[0];
    }

    public override void CreateRole(string roleName)...
    public override bool DeleteRole(string roleName, bool throwOnPop
    public override bool RoleExists(string roleName)...
    public override string[] FindUsersInRole(string roleName, string
```

SQL Server Security

- Authentication:
  - Windows Authentication (`"Integrated Security=sspi; ..."`)
  - SQL Server Authentication (`"User ID=sa; Password=; ..."`)
- Authorization:
  - always managed by SQL Server even if using Windows authentication.



References

- Security Overview: http://msdn.microsoft.com/en-us/library/hdb58b2f.aspx