M13 L1
Security and Reliability Concepts

Ira A. Fulton Schools of Engineering
Arizona State University

Lecture Outline

Dependability

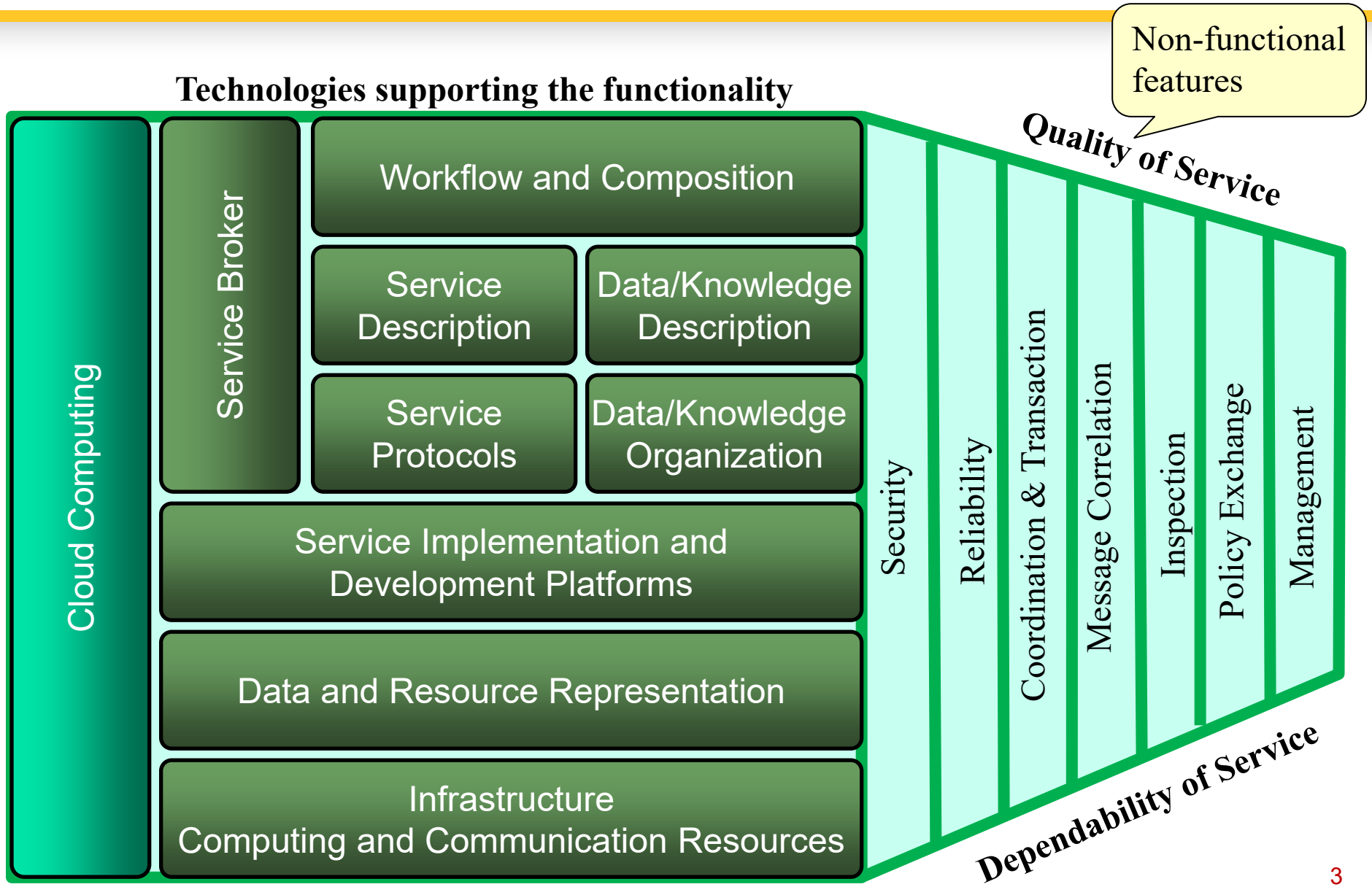
- Reliability
- Availability
- Security

Web Application's Security

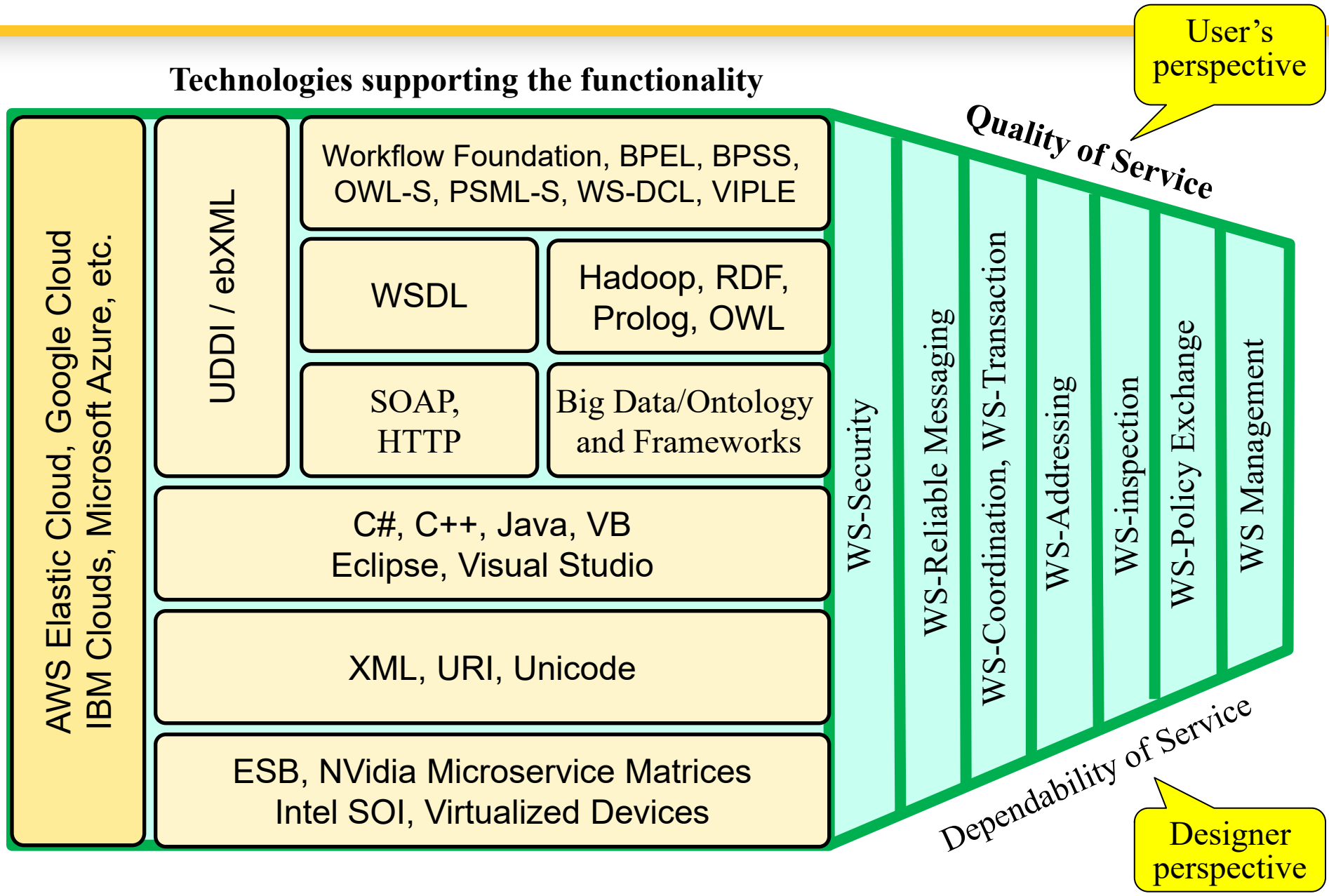
Cryptography Systems

- Secret Key System
- Open Key System

Organization of SOC-Enabling Technologies



Instances of the SOC-Enabling Technologies



Quality of Service (QoS)

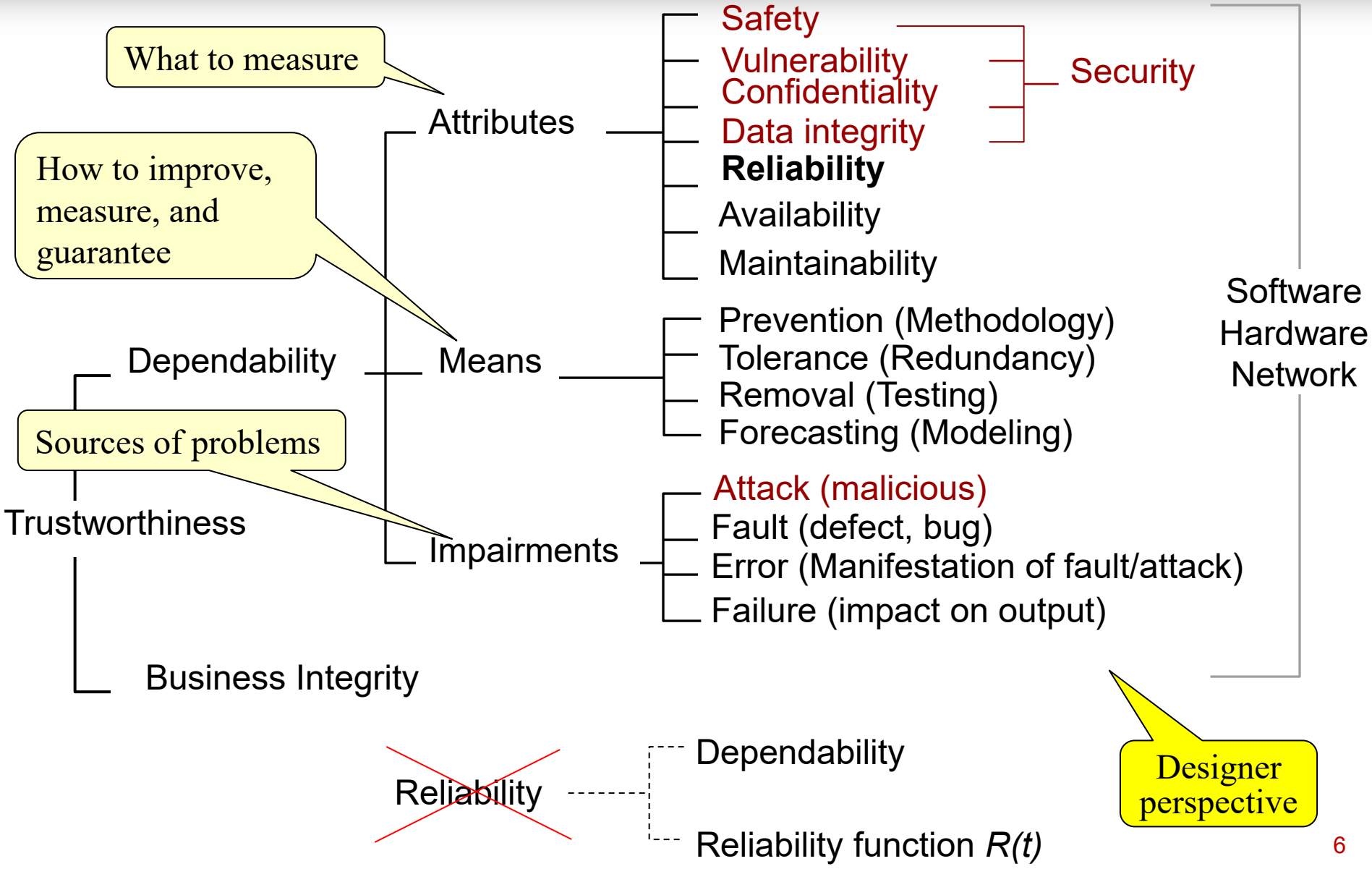
User's perspective

- ISO 8402 [ISO 1986] defines quality as "the totality of features and characteristics of a product or service that bears on its ability to meet a stated or implied need".
- Different fields have different measurable interpretations:
  - Network quality: represents the transmission rates, error rates, and other characteristics that can be measured, improved, and to some extent, guaranteed in advance.
  - Software quality: is the degree to which software conforms to quality criteria. Quality criteria, some are measurable and some are not, include:

    Economy, Correctness, Resilience, Integrity, Reliability,

    Usability, Modifiability, Clarity, Understandability, Validity, Maintainability, Flexibility, Generality, Portability, Interoperability, Testability, Efficiency, Modularity, Reusability, …

Dependability Concept and Terminology



Measurable Failure Probability and Reliability Function

- Let T be a random variable that represents the time interval from time 0 to the time t of the **first failure's** occurrence.
- The probability function: $F(t) = \text{Prob}[T \leq t]$
  is called **failure probability** in the time interval $[0, t]$.
- **Reliability** ensures continuity of service in [0, t].
  $R(t)$ function of a system is the probability that the system has survived in the time interval $[0, t]$, given that it is operational at time 0.
  $R(t) = 1 - F(t) = 1 - \text{Prob}[T \leq t] = \text{Prob}[T > t]$
- **Availability** ensures the readiness of service at time point t.
  $A(t)$ function of a system is the probability that the system is working at time t.

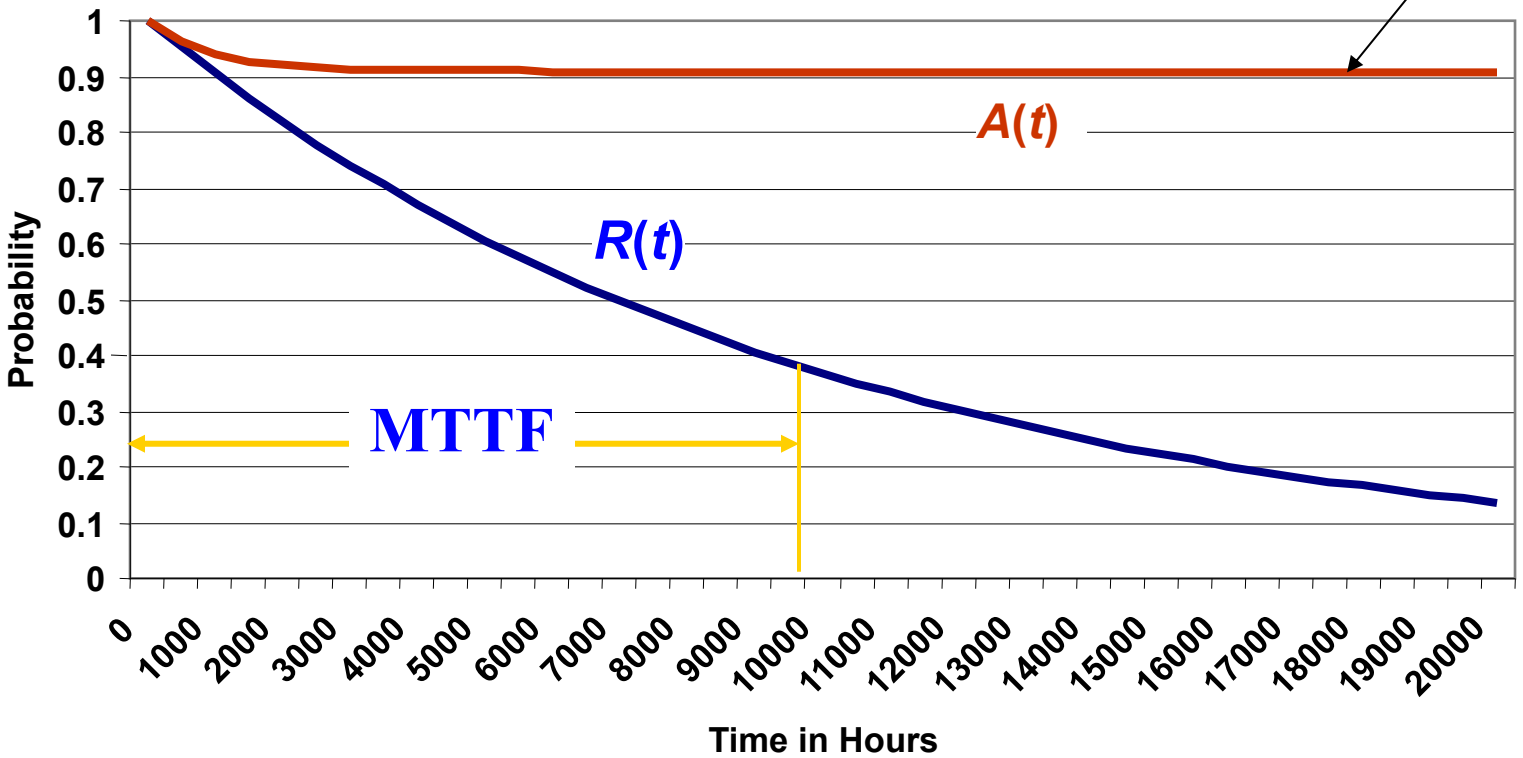
Measurable R(t), A(t) and MTTF (Mean Time To Failure)

For a failure rate $\lambda = 10^{-4}$/hour, $R(t) = e^{-\lambda t}$, MTTF $= 1/\lambda = 10^4$ hours

For a repair rate $\mu = 10^{-3}$/hour, $A(t) = \frac{\mu}{\lambda+\mu} + \frac{\lambda}{\lambda+\mu} e^{-(\lambda+\mu)t}$

If $\mu = 0$, $R(t) = A(t)$, and if $t \rightarrow \infty$, $A(t) = \frac{\mu}{\lambda+\mu}$

$A = \frac{\mu}{\lambda+\mu}$



Security: Vulnerability, Confidentiality, Integrity, and Safety

- **Vulnerability** describes a problem or weakness, such as a programming error or common misconfiguration, that allows a system to be attacked or broken into.
- **Confidentiality** ensures that information is accessible only to those authorized to have access.
- Data/Message **integrity** refers to the validity and consistence of data/message. We also have a concept **provenance** for the traceability of data – where and how does the data come from?
- **Safety** ensures non-occurrence of catastrophic consequence on the environment (human life lost, economic impact, etc.)
- **Security** is the umbrella concept of the above four.

  Security deals with malicious attacks, while **reliability** and **availability** deal with faults, errors, and failures caused by imperfect development and operation environment.

Next:

Web Application Security

Web Attacks, Security Attributes, and Means,


Ira A. Fulton Schools of
Engineering
Arizona State University

M13 L2
Web Application Security and Cryptography

Ira A. Fulton Schools of Engineering
Arizona State University

Lecture Outline

- **Web Application's Security**
- **Web Attacks, Security Attributes, and Means**
- **Cryptography Systems**
  - Secret Key System
  - Open Key System

Common Attacks on the Web

What Web computing model can cause this attack?

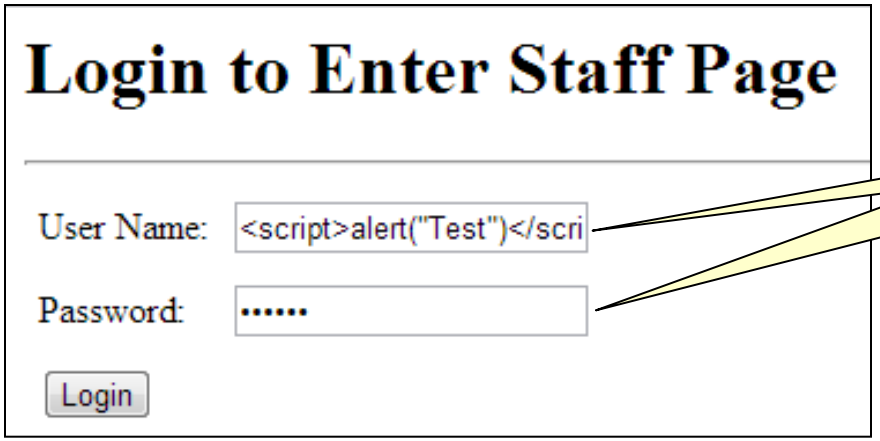
What technique can prevent this attack?

Attack	Description
Cross-site scripting (XSS)	Script programs are used as input data and are executed browser on client side.
Denial of service (DoS)	The attacker floods the network with fake requests, overloading the system and blocking regular traffic.
Eavesdropping	The attacker uses a sniffer to read unencrypted network packets as they are transported on the network
Rootkit and hidden-field tampering	The attacker compromises unchecked (and trusted) hidden fields stuffed with sensitive data in the Web forms
One-click	Malicious HTTP posts are sent via script: Do not click on suspicious links!
Session hijacking	The attacker guesses, phishes, or steals a valid session ID and connects over another user's session.
Database query / SQL injection	The attacker uses a malicious input string, e.g., a usename, that happens to form dangerous SQL commands, or an illegal character that triggers an unexpected operation. When it is sent to the DB for verification, it becomes a command.

Strong password

Cross-Site Scripting (XSS) Attack Example

- A script is used as input data, which can be executed, if not programmed properly, in the browser.



Script code is entered here

If the error message: The username cannot contain special character, then …

View Page Source: If the input check is done in client-side script, modify the script to turn off the input check.

Another way of adding the script is to intercept the SOAP message and inject script into SOAP body.

Create a new page based on the modified page source, and enter script <script> … </script> again.

SQL Injection Attack

- Similar to XSS attack, an attacker enters username and password that contains SQL commands, such as SELECT, CREATE, UPDATE, DELETE, and INSERT.
- When the system sends the username and password to the database to check if the username and password exist, the SQL commands are sent and executed instead.

Security Means and Attributes

- **Authentication** requires an entity to present proof of identity and verify: *Are you who you say you are?*
- **Authorization** confirms that an entity is entitled to access a resource: *What are you allowed to do, after you are confirmed who you are?*
- **Cryptography** for
  - **Message confidentiality** ensures that only the intended recipient of a message can read the message in its original form.
  - **Message/Data integrity** ensures that a message is not tampered with during transit or during storage.

A Few Basic Concepts in Security

- **Credential**: Evidence or information that provides proof of identity, such as a birth certificate or business license. A credential contains a claim, which provides information about a person or an **entity**, such as an account name or an e-mail address. Claims are certified by a trusted party called an **issuer**. A credential can be protected by a water mark, a secret code, etc.
- **Entity**: Something that possesses credentials, such as a person, an organization, a computer, or a session.
- **Resource**: Something an entity might want to access, such as a page, a file, a database, a service, or an operation.
- **Security token**: A representation of a credential that can be passed as part of a message, such as username and password, encrypted or not.

Cryptography Systems

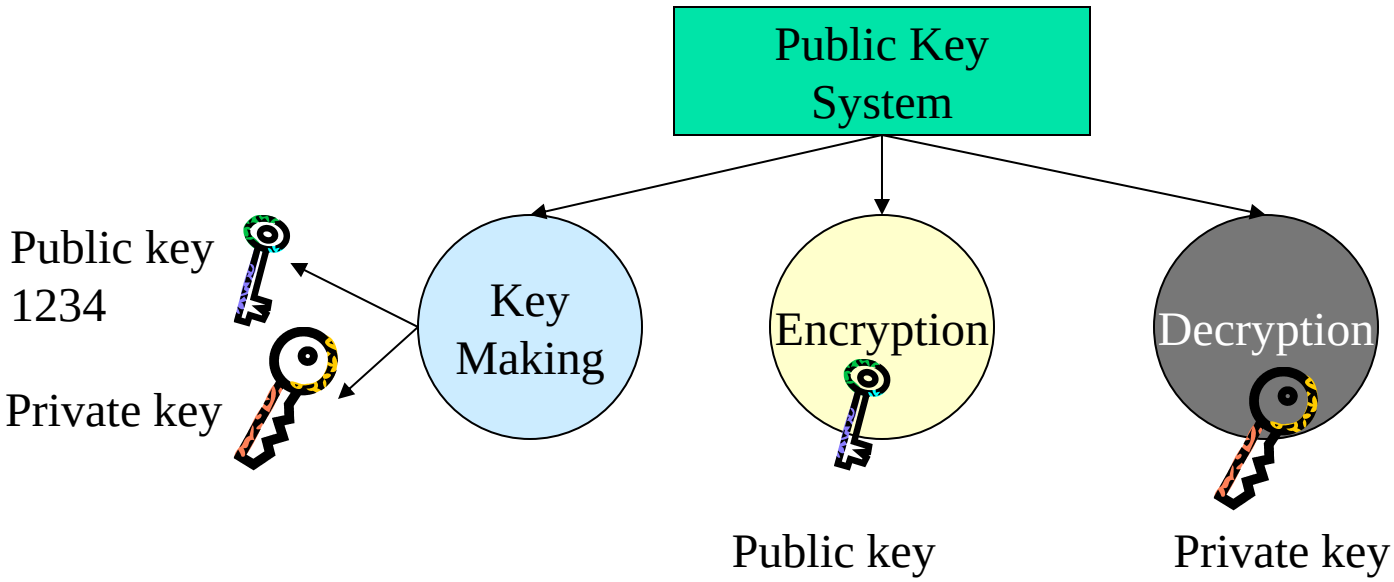
- **Secret key systems**: A secrete key is used to encrypt the data/message
  - For example: cipher encryption adds an integer (key) to each character;
  - Problem: How can you safely transfer the key to the receiver through Web/internet?
- **Public Key Infrastructure**: two keys are created.
  - Encryption for data confidentiality: An open key E for encoding, and a private key D for decoding.
    Reverse: It is exponentially complex to obtain D from E
  - Digital signature: An open key D for decoding and a private key for encoding.
    Reverse: It is exponentially complex to obtain E from D.
  - Other reverse example: Solving an equation system can be complex, but verify a solution is simple.

Analogy of Public Key Infrastructure

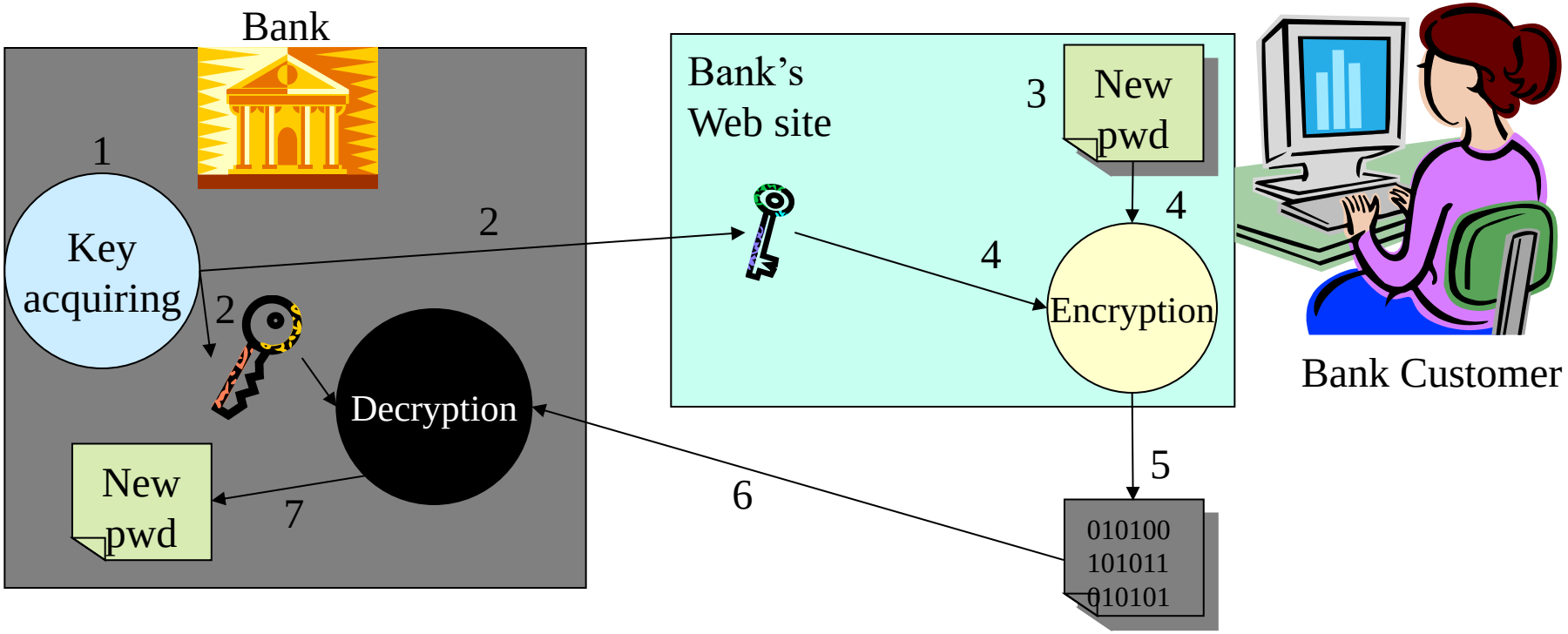
Consider a special **safe box**, which requires a key to open and a different key to lock.

- Data confidentiality: You want to send a confidential document to your bank. You order a safe box from the bank, with a public key for locking. The key is printed on the box: 1234. You put the document in the safe box. Lock the safe box using key 1234, and mail the safe box to the bank. Only the bank has the key to open the safe.
- Digital signature: The government sends an important announcement to the citizens, e.g., evacuation. The government puts the announcement in the box, locks it using a private key, and sends it to the citizens.
  The unlocking key is open and printed on the box:1234.

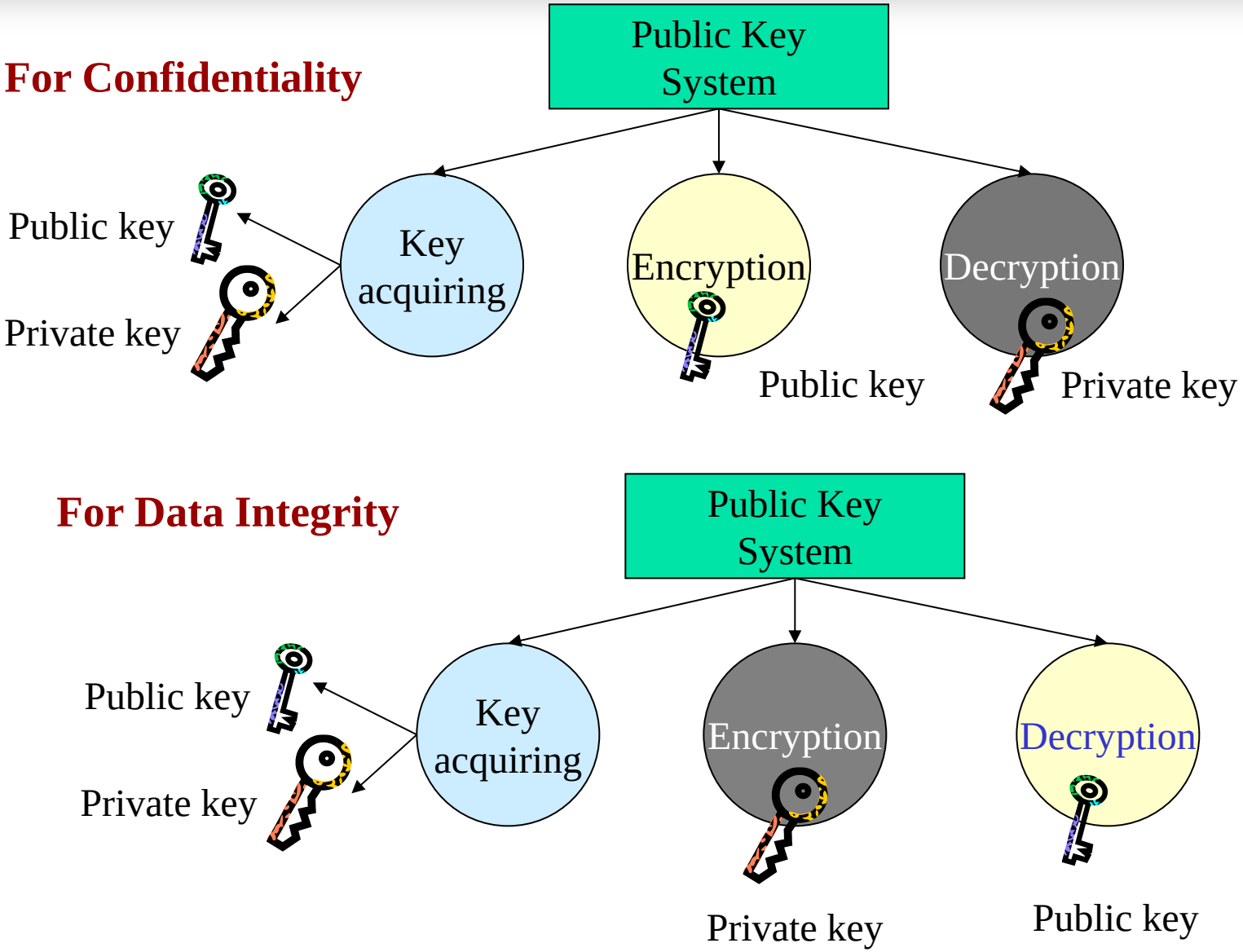
Public Key Encryption for **Confidentiality**



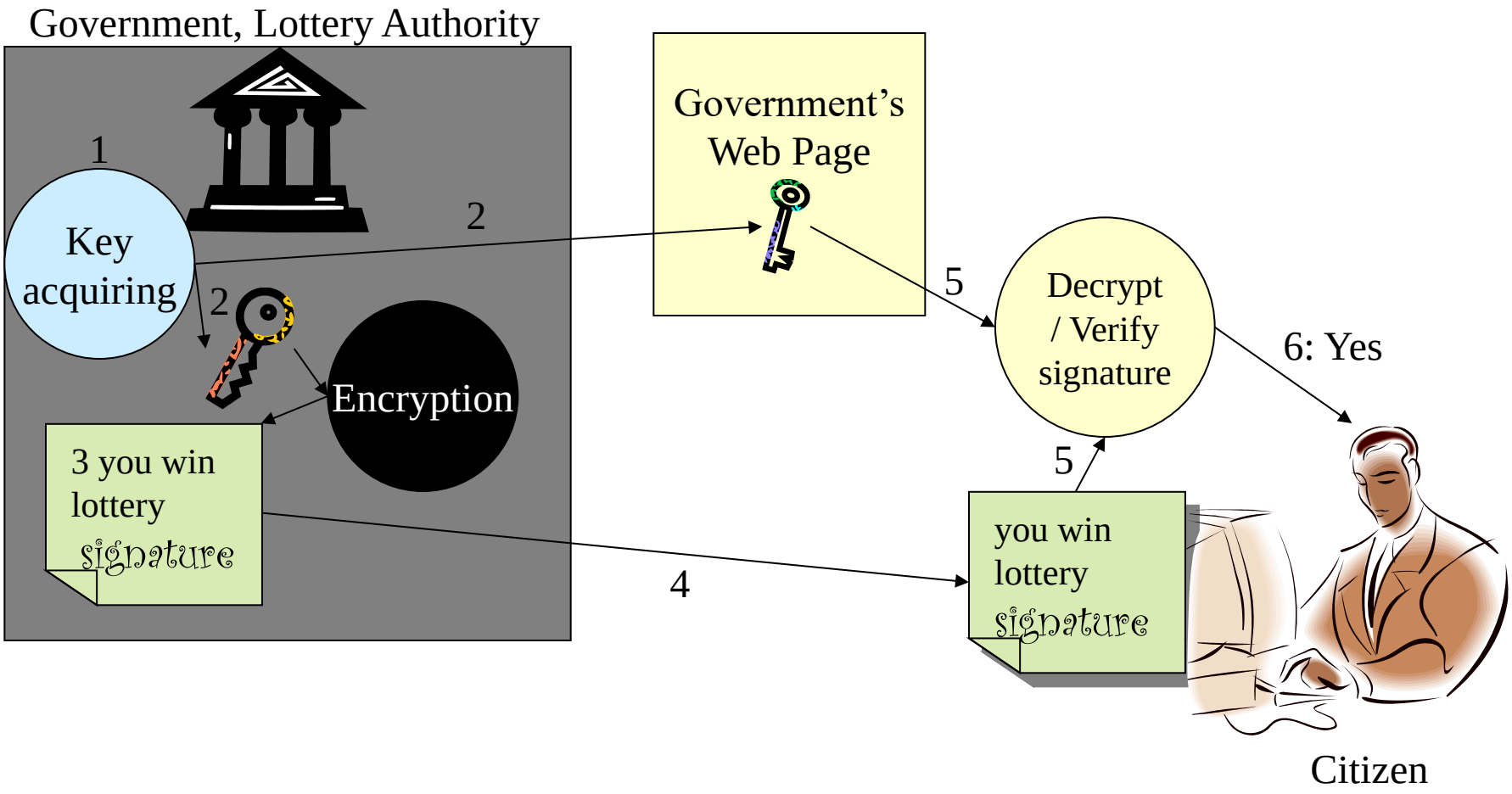
Example: Public Key Encryption for **Confidentiality**



Public Key Encryption for Integrity -- Digital Signature



Example: Public Key Encryption for **Integrity**




Ira A. Fulton Schools of
Engineering
Arizona State University

M13 L3
IIS and Windows-Based Security Systems

Ira A. Fulton Schools of Engineering
Arizona State University

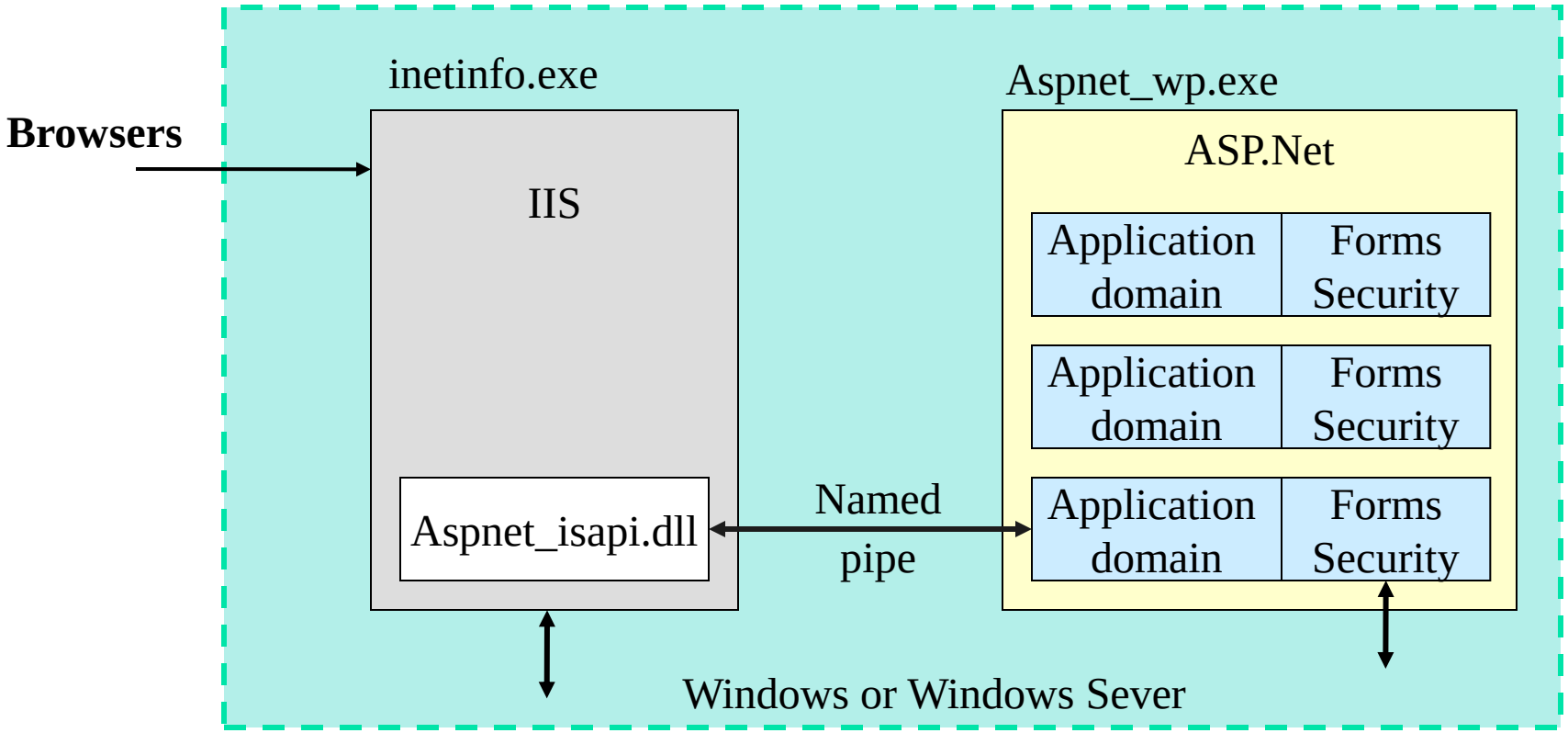
Lecture Outline

- IIS Supported Web Security
- Using Windows Security System for Web Security

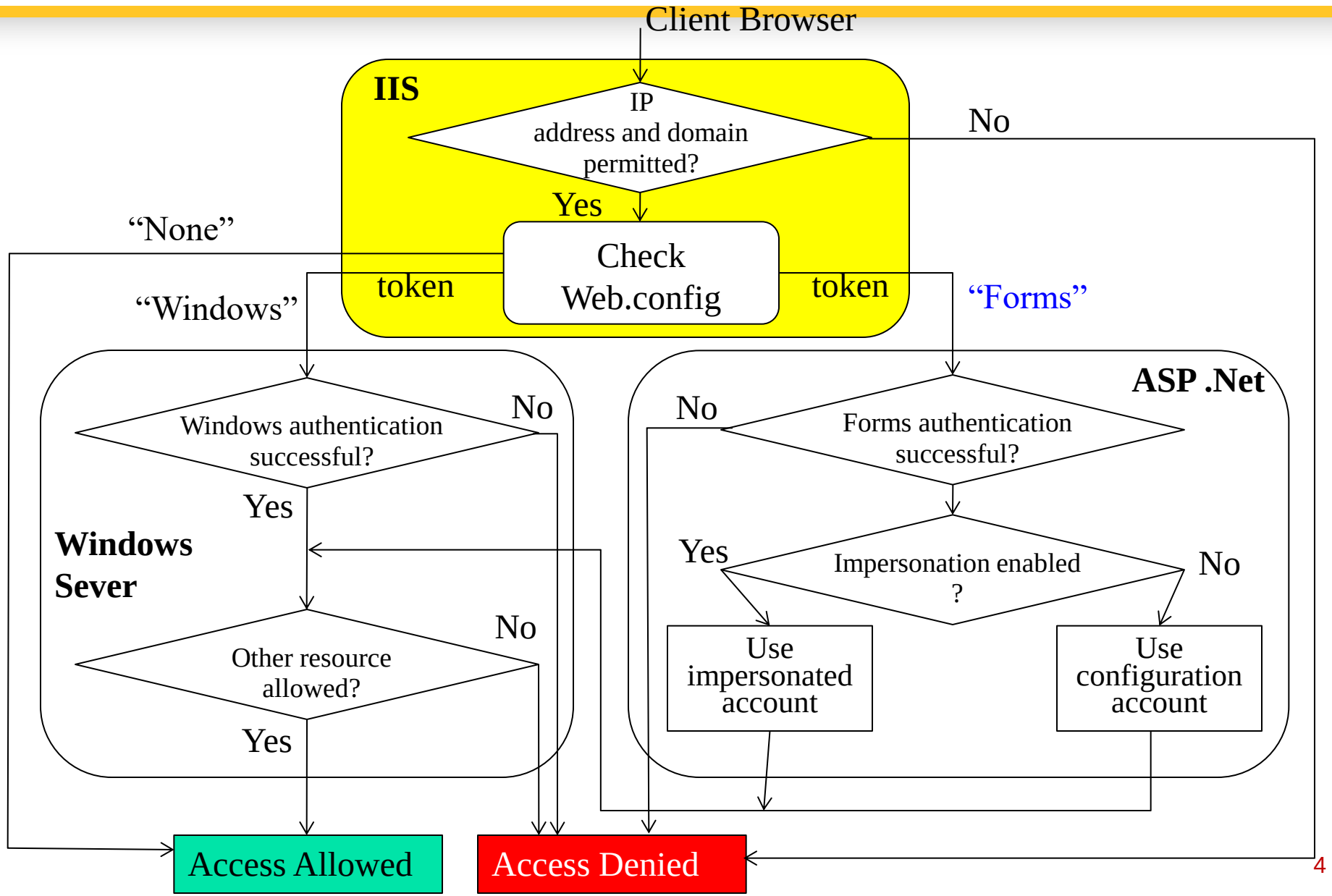
Security Mechanisms Related to ASP .Net

- IIS Security Mechanisms
- Windows Security Mechanisms
- ASP .Net User-Defined **Forms Security** Mechanisms

Next lecture



Security Checks

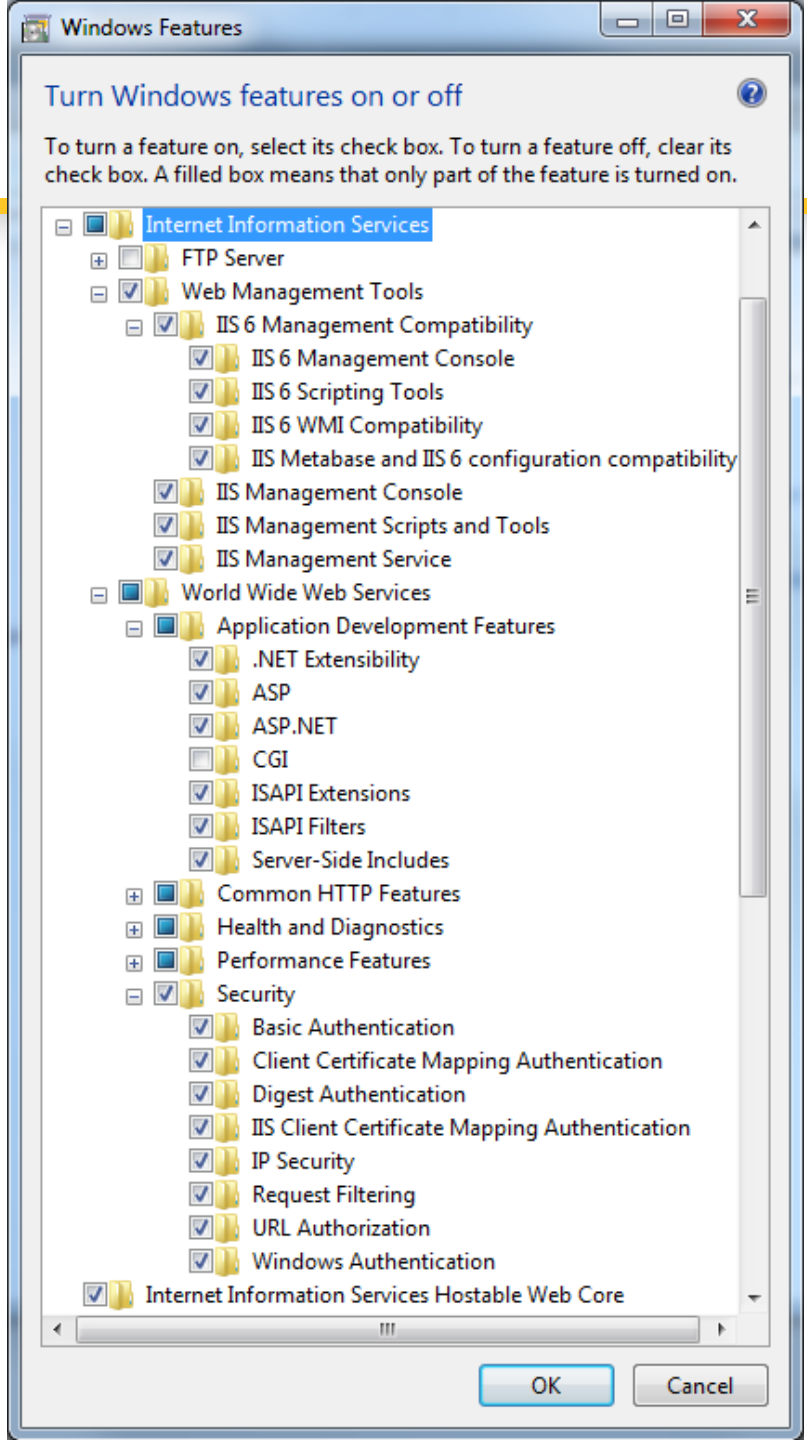


IIS Security

- Web applications are deployed in virtual directories that are URL-addressable on the server. Remote clients can't access files outside the virtual directories.
- IIS supports IP address and domain name restrictions, enabling requests to be granted and denied based on the IP address or domain of the requestor.
- IIS supports encrypted HTTPS connections using the Secure Sockets Layer (SSL) family of protocols.
  - SSL doesn't protect resources on the server,
  - It prevents eavesdropping on conversations (communication) between Web servers and remote clients.
  - HTTPS implements SSL at HTTP and the layers below.
- IIS creates for every request an access token (security token).
  - The access token enables ASP.Net or the Windows/sever or ASP .Net Forms to perform ACL (Access Control List) checks on resources targeted by the request.
  - Each file can be given an access group (anonymous or with credential)
  - If the request runs as Bob and Bob isn't allowed to read Staff.aspx, the request will fail when it attempts to read Staff.aspx.
  - IIS makes Bob's access token available to OS or ASP.NET so that ASP.NET can perform access checks of its own.

Turn on IIS

- Turn on your IIS:
  Control Panel
  → Programs
  → Programs and Features
  → Turn Windows Features on or off



Using Windows Security System for Web Security
A Case Study

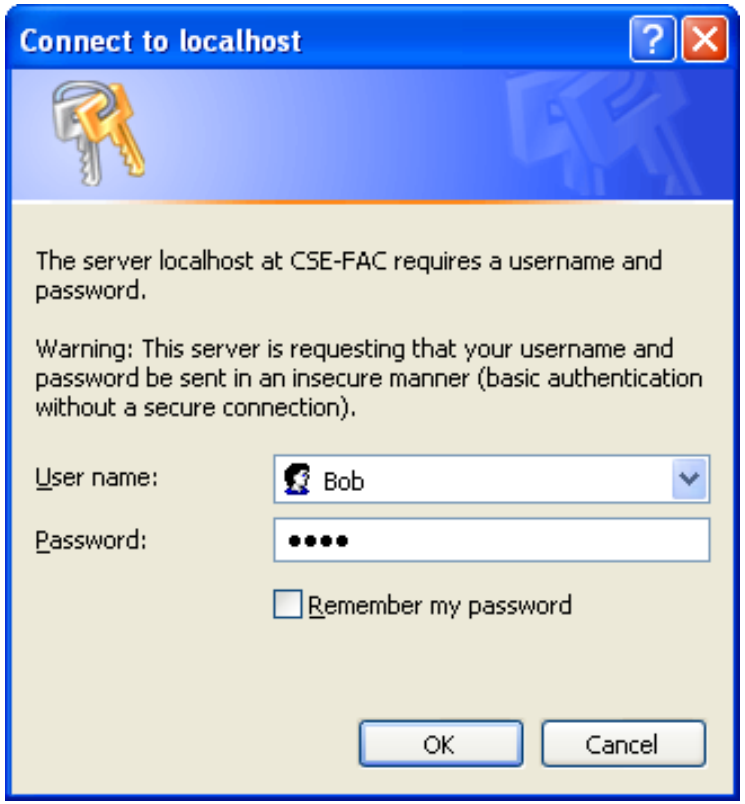
Combining ASP .Net (Web.config),
IIS (as the entry point, always needed), and
Windows Security Mechanisms

Why do we want to use Operating System's Security

- Operating system's authentication is widely used in corporate network or intranet access.
- No programming is needed. Just pass the security token to the OS.
- If the user has the credential for the OS, the user can access the Web application.
- It is useful for employees to access the internal Web applications.

Using Windows Security

- Anonymous: No access control (typically disabled)
- Windows authentication: Use Windows login credential to authenticate users
  - Basic authentication
    - Use username and password to authenticate users
    - Password is sent in clear text
  - Digest authentication
    - Password is sent in encrypted text



Case Study: Corporate Network Security

Source: Jeff Prosise, Programming Microsoft .Net, Core Reference, Microsoft Press.

- CorpNet: A Web Application models a simple corporate Intranet application;
- It uses Windows authentication and program the authorization in C# to restrict access;
- It consists of the following pages & files
  - *General.aspx*, which provides general information about the company; Anyone can see this page.
  - *Salaries.aspx*, which lists the salary of the employee; Restricted access.
  - *Duties.aspx*, which accesses the Duties.xml file; Restricted access.
  - *Duties.xml*, which lists the current employee duties. Restricted access to outsider.
  - *Web.config* file

General.aspx File

```
<%@ Page Language="C#" %>
<html>
  <body>
    <h1>Welcome to CorpNet!</h1>
    <hr>
     Welcome to the corporate intranet! We know who you are
     because you had to provide a username and password to see
     this page. To prove it, your username is shown below.<br>
    <h3>
      <%
        if (User.Identity.IsAuthenticated)
            Response.Write (User.Identity.Name);
      %>
    </h3>
  </body>
</html>
```

Built-in functions to support:
Call Windows authorization mechanism based on the Windows accounts.
If true, print username.

Salaries.aspx File: Authorization Control

```
<%@ Page Language="C#" %>
<html>
  <body> <h1>Salaries</h1>    <hr>
    <%
      if (!User.Identity.IsAuthenticated)
          Response.Write ("Sorry, but no salary information " +
              "is available for unauthenticated users.");
      else {
          if (User.Identity.Name.IndexOf ("Jeff") != -1)
                          Response.Write ("Jeff's salary is $650,000.");
          else if (User.Identity.Name.IndexOf ("Ryan") != -1)
                          Response.Write ("Ryan's salary is $60,000.");
          else if (User.Identity.Name.IndexOf ("Bob") != -1)
                          Response.Write ("Bob's salary is $30,000.");
          else if (User.Identity.Name.IndexOf ("Julia") != -1)
                          Response.Write ("Julia's salary is $70,000.");
          else if (User.Identity.Name.IndexOf ("Mary") != -1)
                          Response.Write ("Mary's salary is $45,000.");
          else Response.Write ("No salary information is available for " + User.Identity.Name);
      }
    %>
  </body>
</html>
```

To test the program, you may change one of the names, e.g., jeff, to your own login name on your computer. When the login window pops up, you can type your own login name and password

Are we using policy-based computing?

How can we make it policy-based computing?

Built-in functions to support:
Use this piece of C# code to define the access privilege (authorization) of each user.

Duties.aspx File

```
<%@ Import Namespace="System.Data" %>
<html>
  <body>
    <asp:DataGrid ID="MyDataGrid" Width="40%" RunAt="server" />
    <asp:Label ID="Output" RunAt="server" />
  </body>
</html>
<script language="C#" runat="server">
  void Page_Load (Object sender, EventArgs e) {
      try {
          DataSet ds = new DataSet ();
          ds.ReadXml (Server.MapPath ("Duties.xml"));
          MyDataGrid.DataSource = ds;
          MyDataGrid.DataBind ();
      }
      catch (Exception) {
          Output.Text =
              "An error occurred processing this page";
      }
  }
</script>
```

Load the XML file into a DataSet structure, which is a set of tables.

See server control

If impersonate in Web.config is properly set, unauthenticated user cannot access the file even through a program

Duties.xml

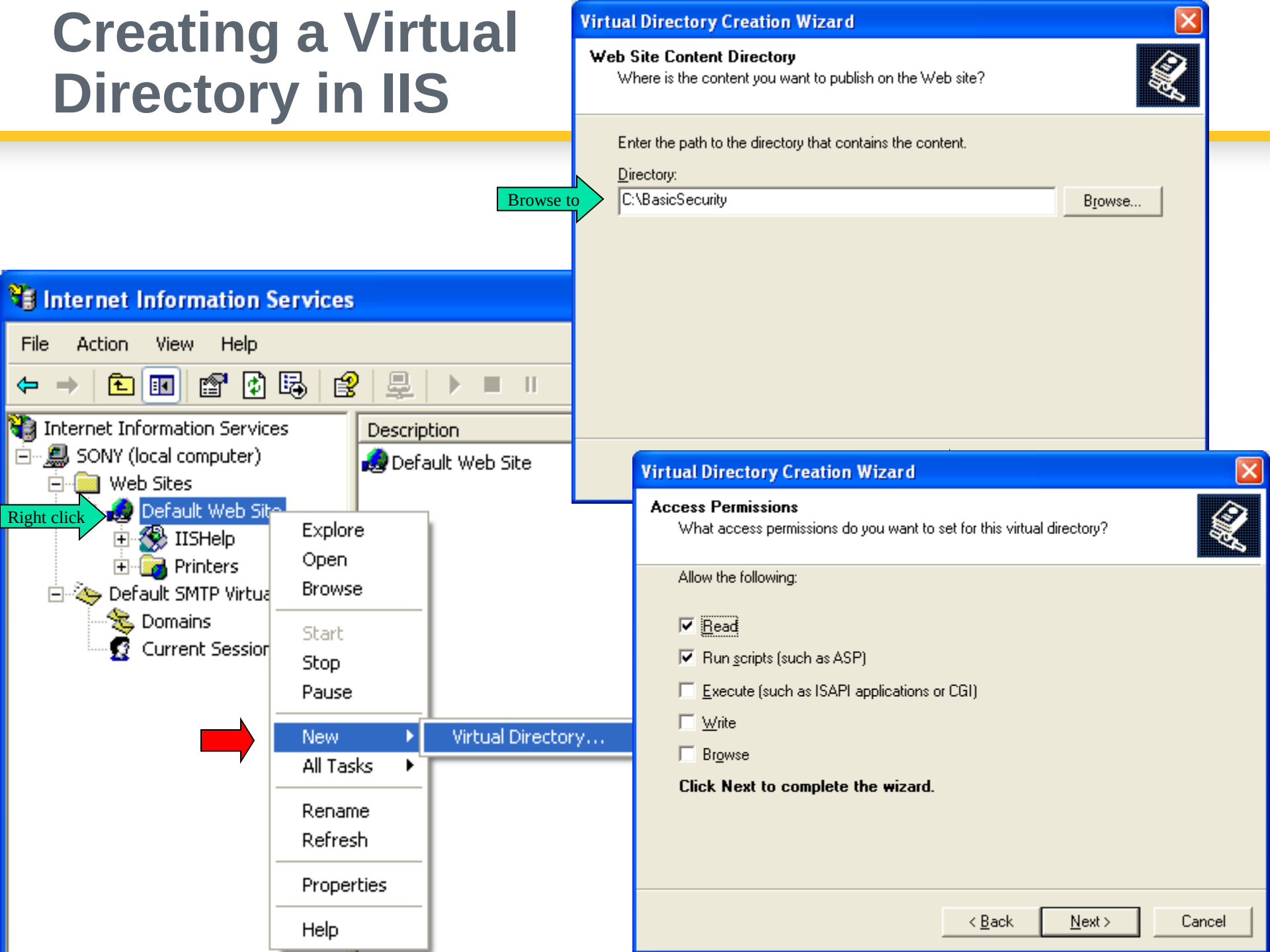
```
<?xml version="1.0" encoding="UTF-8" ?>
   <Duties>
        <Duty>
             <Name>Julia</Name>
             <Portfolio>Security Manager</Portfolio>
        </Duty>
        <Duty>
             <Name>Bob</Name>
             <Portfolio>Contractor Manager</Portfolio>
        </Duty>
        <Duty>
             <Name>Jeff</Name>
             <Portfolio>Architect</Portfolio>
        </Duty>
        <Duty>
             <Name>Ryan</Name>
             <Portfolio>International Collaboration</Portfolio>
        </Duty>
        <Duty>
        <Name>Mary</Name>
        <Portfolio>Business Manager</Portfolio>
   </Duty>
</Duties>
```

In this implementation, an authenticated user will see all the Duties.

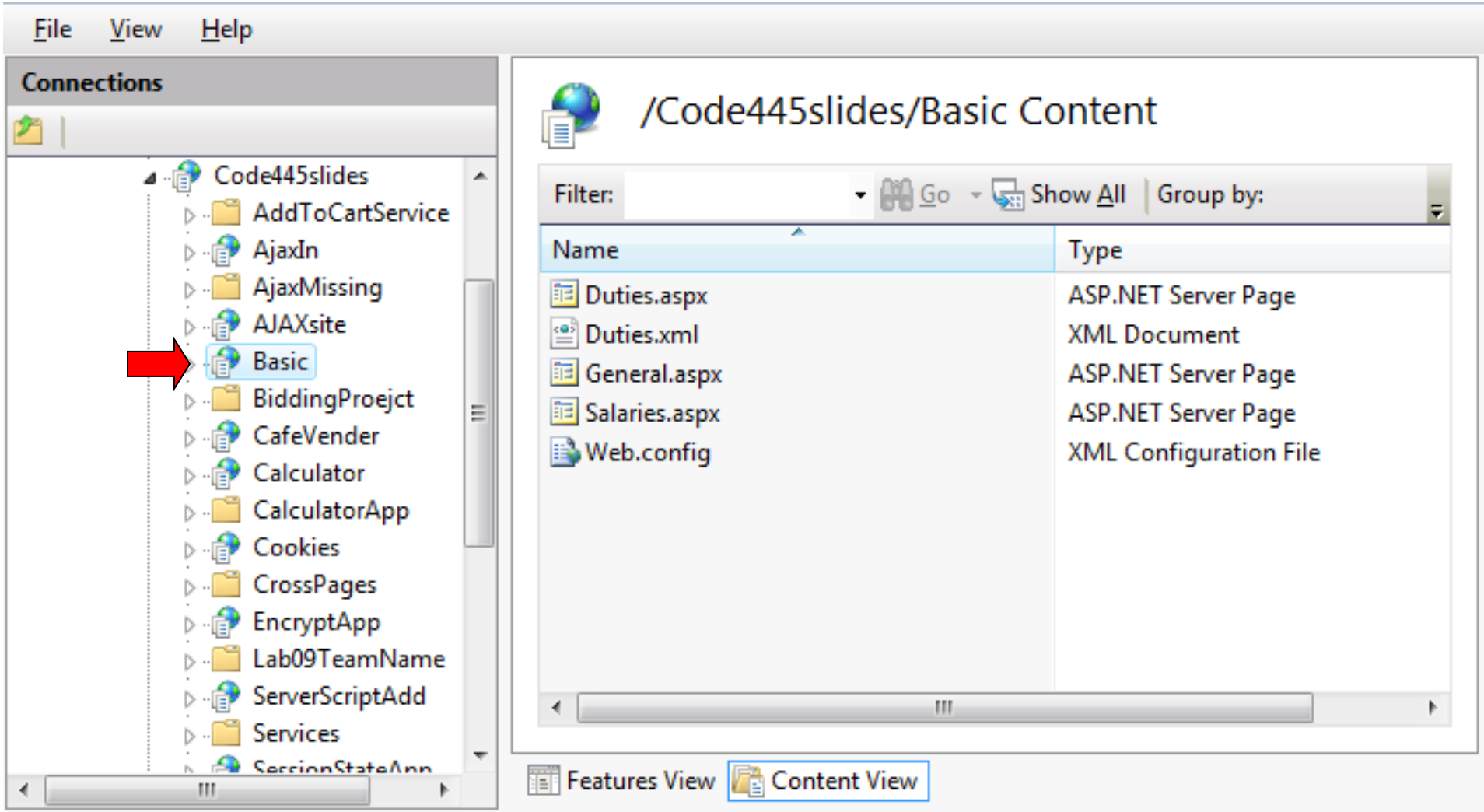
Windows-Based Security Deployment and Testing

1. Create a directory named Basic somewhere — anywhere — on your computer, e.g., C drive.
2. Use the IIS configuration manager to create a virtual directory named “Basic” and link it. How? See next page.
3. While in the IIS configuration manager, configure Basic to require basic authentication and to disallow anonymous access. How?
    1) Right-click Basic in the IIS configuration manager and select Properties from the ensuing context menu.
    2) Go to the Directory Security page of the property sheet that pops up and click the Edit button under “Anonymous access and authentication control.”
    3) In the ensuing dialog box, uncheck “Anonymous access” and check “Basic authentication”.
    4) Right click the folder to convert the folder into application
    5) OK the changes, and then close the configuration manager

Creating a Virtual Directory in IIS

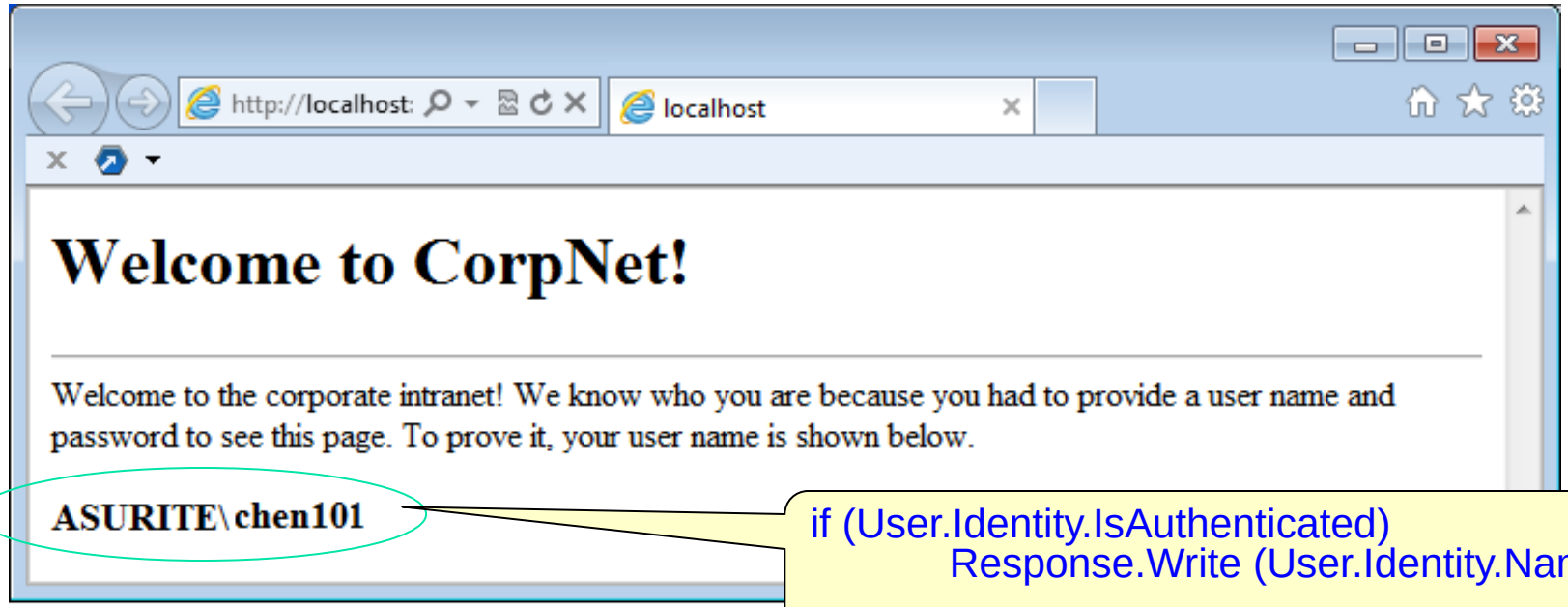


A directory *BasicSecurity* is created in IIS



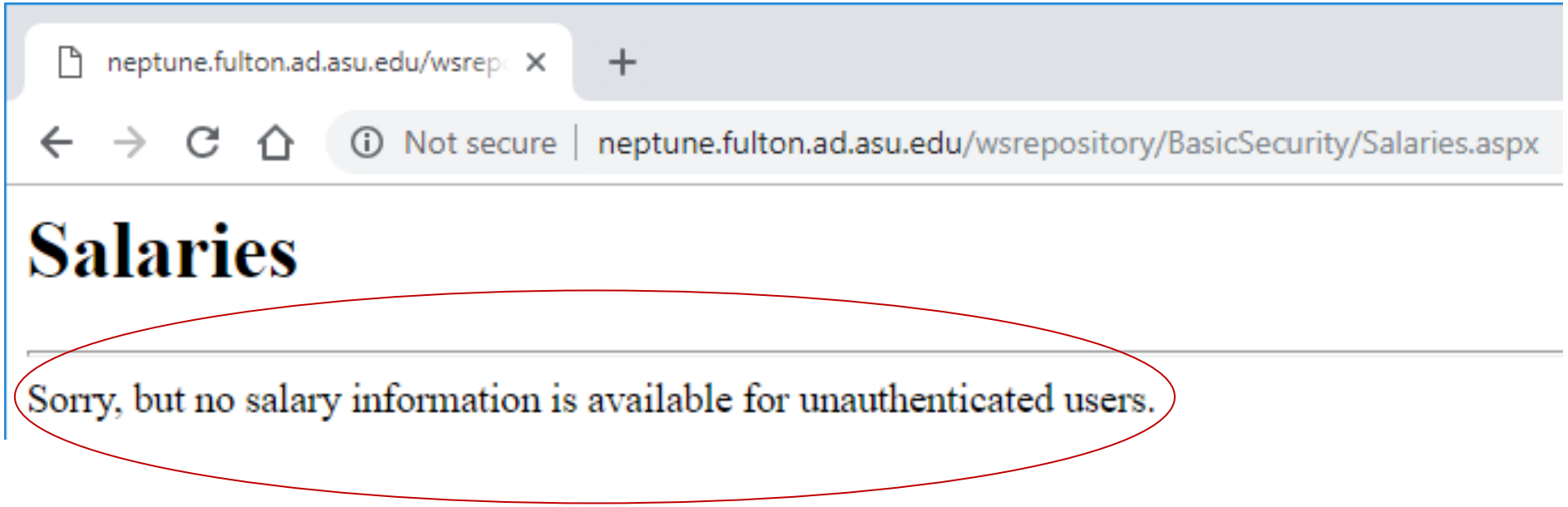
Creating the Authentication (contd.)

4. Create user accounts on your Windows for testing purposes, e.g., “Bob”, “Jeff”, “Julia”, …
5. Change the permissions on *Salaries.aspx* by removing the Anonymous access. See next page
6. Type *http://localhost/Basic/General.aspx* into your browser’s address bar to call up General.aspx. You will see the page, because, by default, it allows anonymous access.

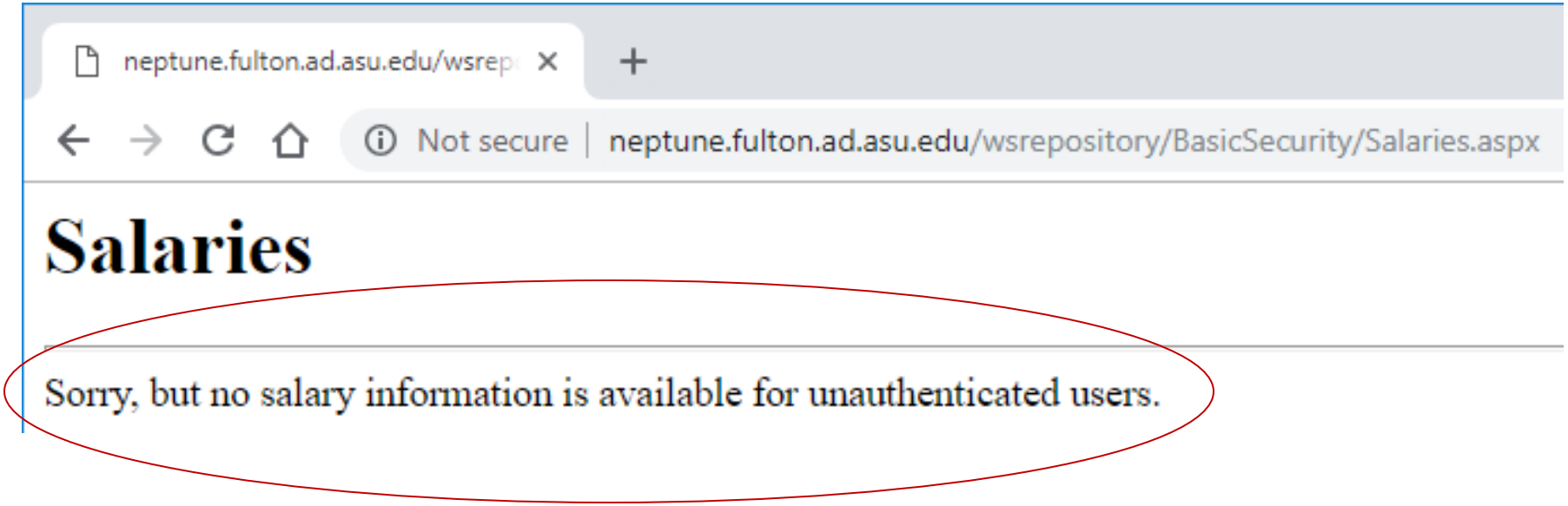


Creating the Authentication (contd.)

7. Type *http://localhost/Basic/Salaries.aspx* into your browser's address bar. A dialog box will pop up, because the Salaries.aspx requires callers to be authenticated using basic authentication. Enter Bob's username and password.



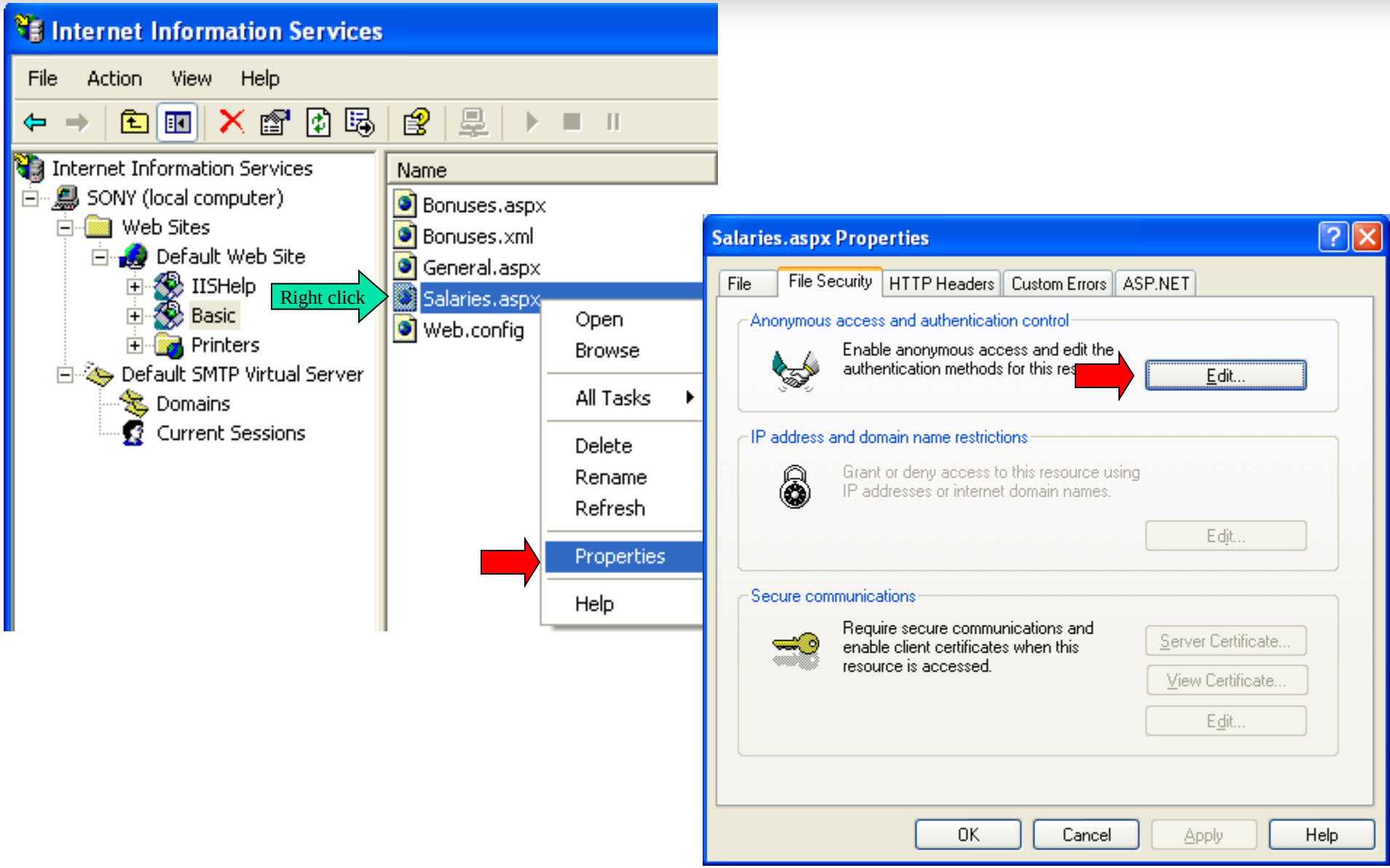
Access Salaries.aspx before changing permission



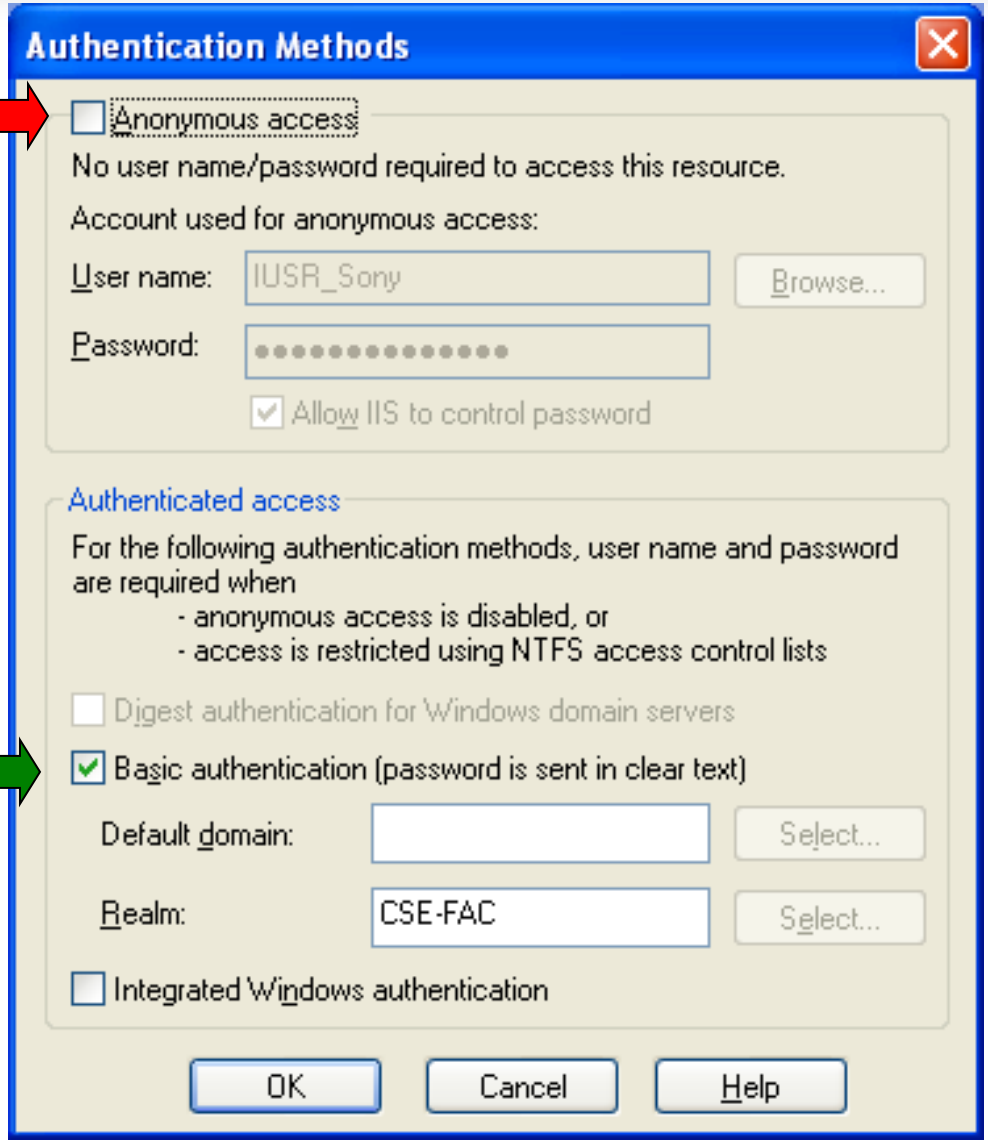
The page is accessible, but the code in the page prevents user from seeing the confidential information:

```
if (!User.Identity.IsAuthenticated)
        Response.Write ("Sorry, but no salary information " +
            "is available for unauthenticated users.");
    else { …
```

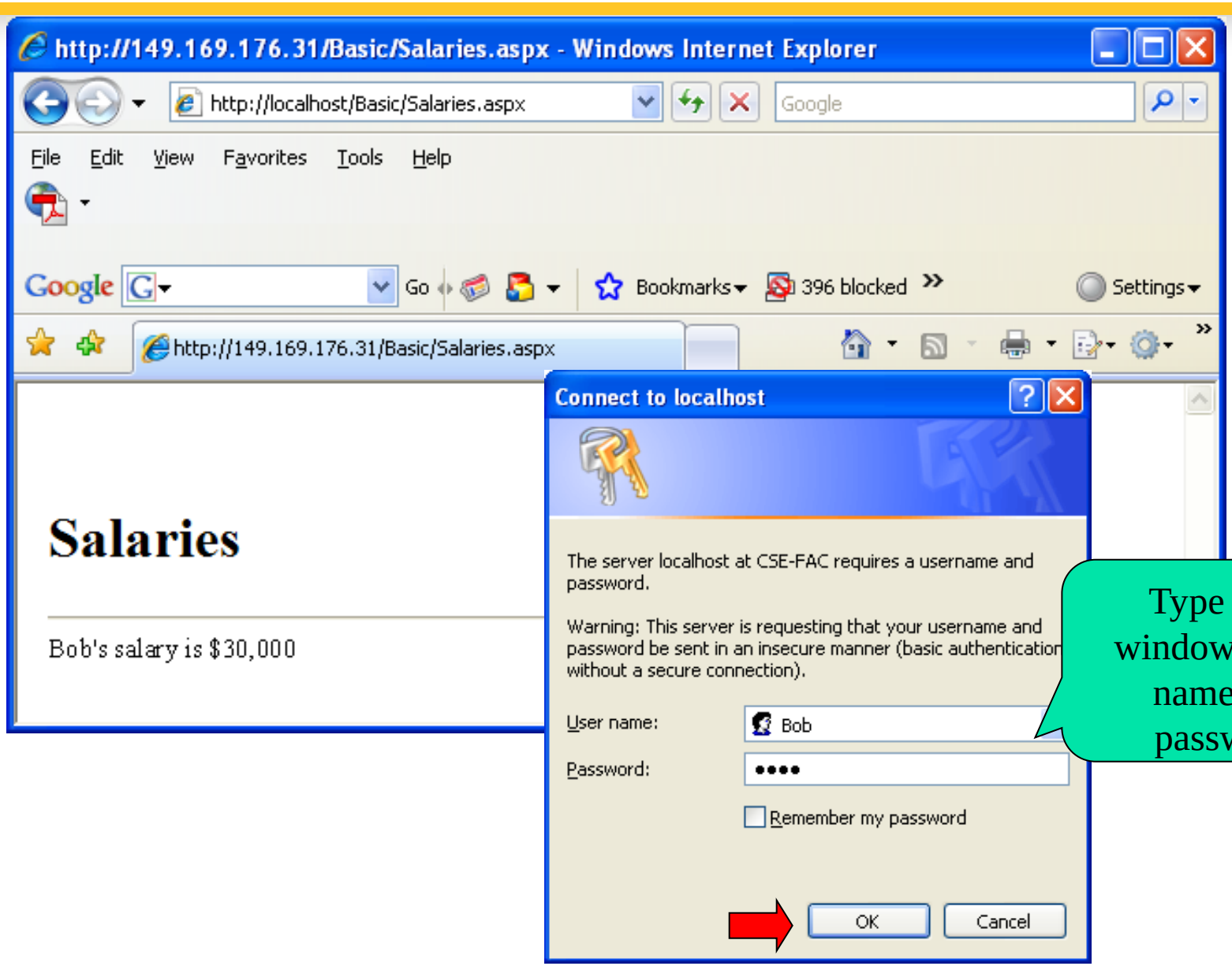
Change the Permission to Basic Security



Remove Anonymous access



Authenticated Access




Ira A. Fulton Schools of
Engineering
Arizona State University

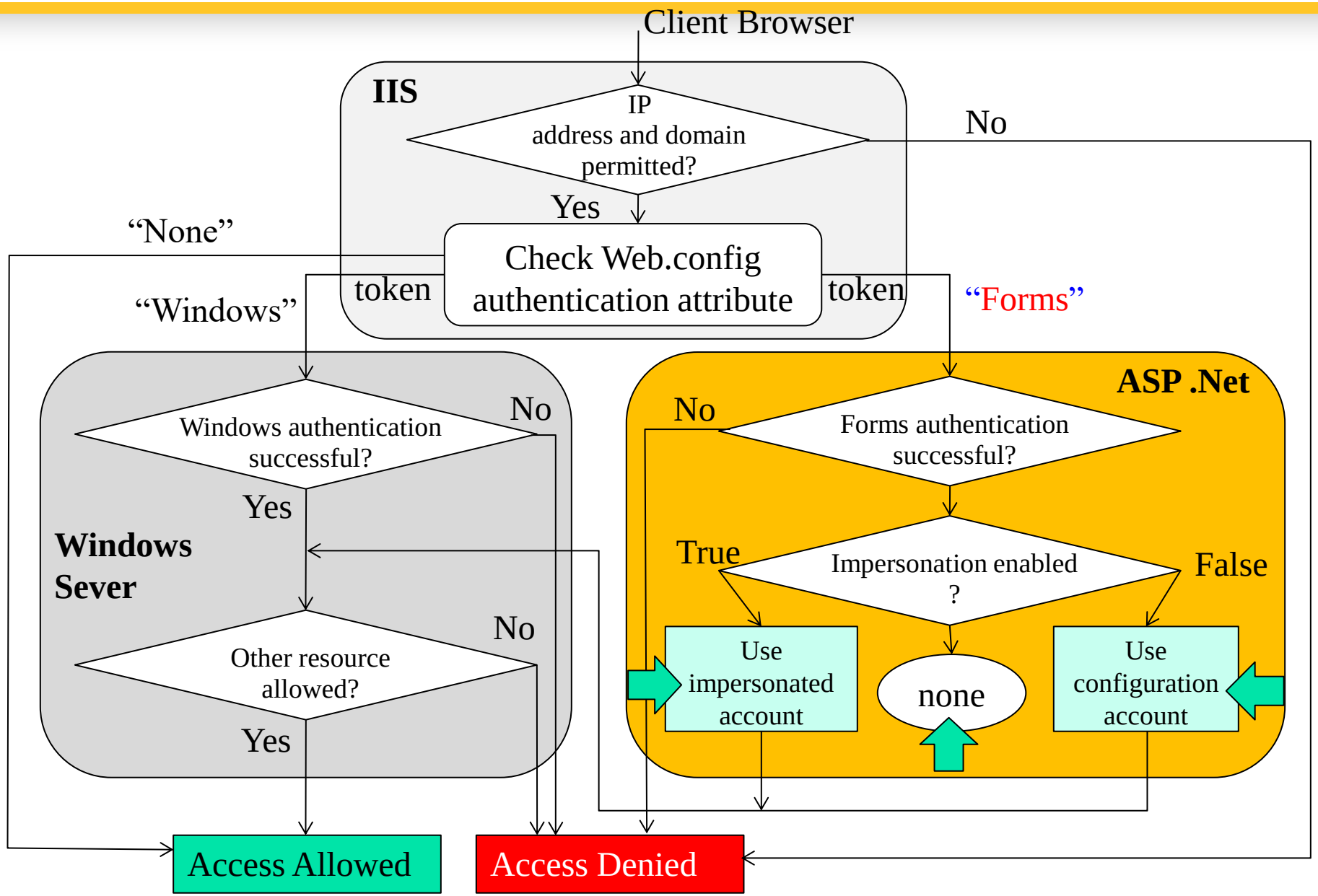
M13 L4
Forms Security Concepts

Ira A. Fulton Schools of Engineering
Arizona State University

Lecture Outline

- Security Check Overview and Principles
- Forms Security Organization
- Security Related Files and Organization

Security Check Overview



Entry Point Control in Web.config File

```
<configuration>
    <system.web>
        <authentication mode="Forms" />
        <identity impersonate="false"/>
    </system.web>
</configuration>
```

Can be Forms, Windows, or None

Make sure that the same credential coming from the IIS is applied

The identity element's attribute impersonate is used for controlling the access to the resources (files) on the server's hard drive.

Impersonation Disabled

- <identity impersonate = "…"> attribute **not** defined (not given): The default will be applied:
  - The application will inherit the identity of the worker process (aspnet_wp.exe), which runs using an account (defined in machine.config) with weaker privileges than the local system account;
  - By doing so, an intruder will not have the local account access even if security is breached (has administrator's password). It is because Local System account has access to almost all resources on the local computer that are not specifically denied to it;
  - The strongest security option – It may be too strong though:
  - This option may deny a user to access a disk file, e.g., an XML file. In this case, you need to define the attribute (next page).

Impersonation enabled with true or false value

- <identity impersonate = "true" userName="domain\user" password="password" />

  ASP.NET impersonates the token generated using this identity specified in the Web.config file given in the identity element.
  This feature is useful for developers to test the program using a different account.

- <identity impersonate = "false" >
  ASP.NET impersonates the token passed to it by IIS, which is either an authenticated user or the anonymous Internet user account. This is the **common use**. 

Forms Security

- Forms security is a common way of implementing access control in Web applications.
- Simply ask a user to type credentials (typically a username and a password) into a Web form.
- The credentials can be issued
  1) by the administrator, e.g., using a password generator, or
  2) through self-registration
- The credentials can be saved in different places and in different forms on the server side and client side:
  - Web.config, in clear text or encrypted
  - XML or text file
  - Database
  - Browser and cookies on the client side

Forms Security in ASP .Net

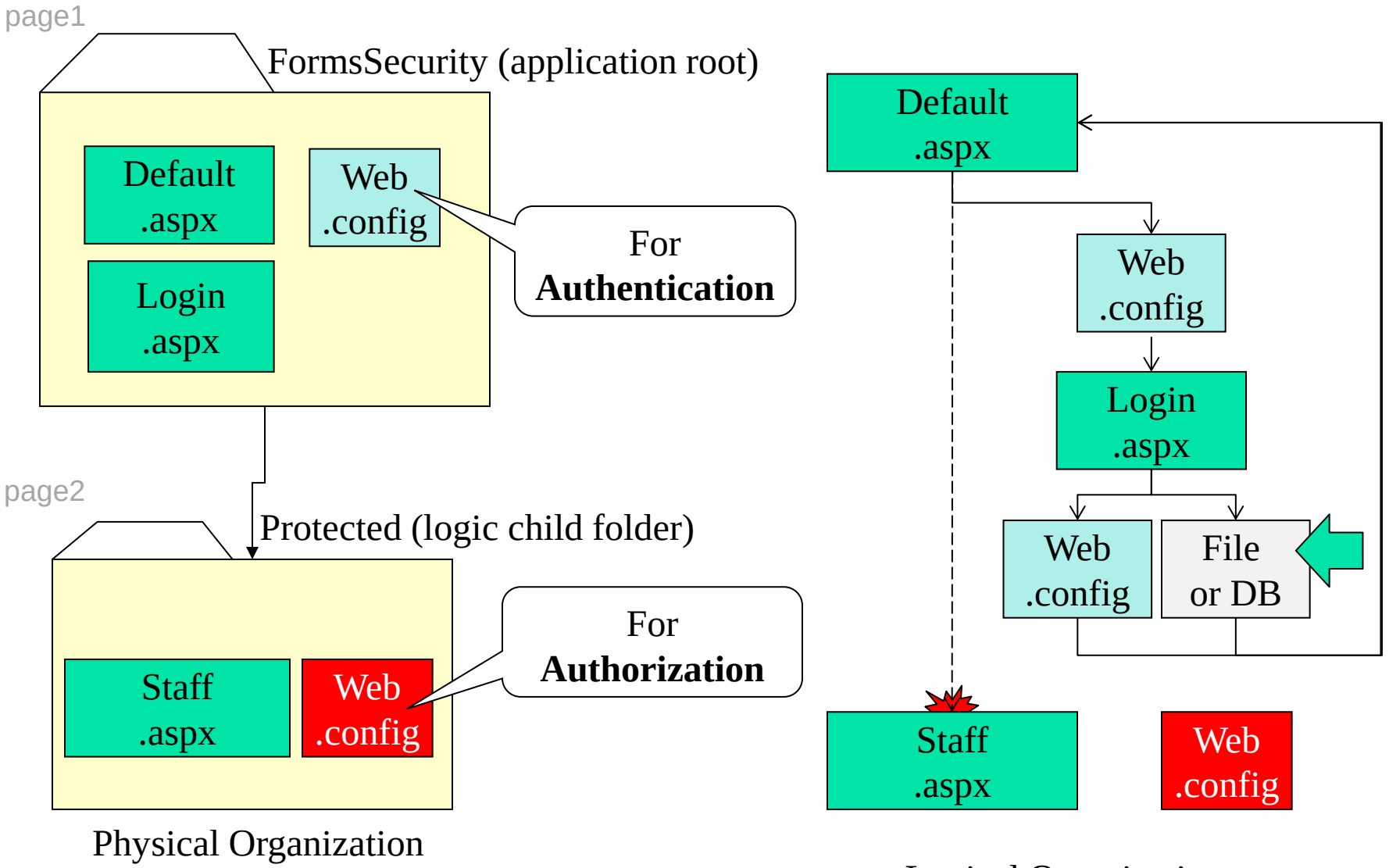
- ASP.NET will save the credentials on server side and can save on client side: Web browser (temporal) or in cookies (persistent).
- When a user attempts to access a protected resource at the first time, ASP.NET automatically redirects the user to the login page.
- If the login is successful, ASP.NET then issues the user an authentication ticket in the form of a cookie, and automatically redirects the user back to the page originally requested.
- The ticket allows that user to revisit protected portions of the site without having to login again and again. One can control the ticket's lifetime and decide how long the login is good for.
- Both authentication and authorization can be defined.
- Example MyASU site: login once and then back and forth
- → Learn by examples

Files of Forms Authentication/Authorization

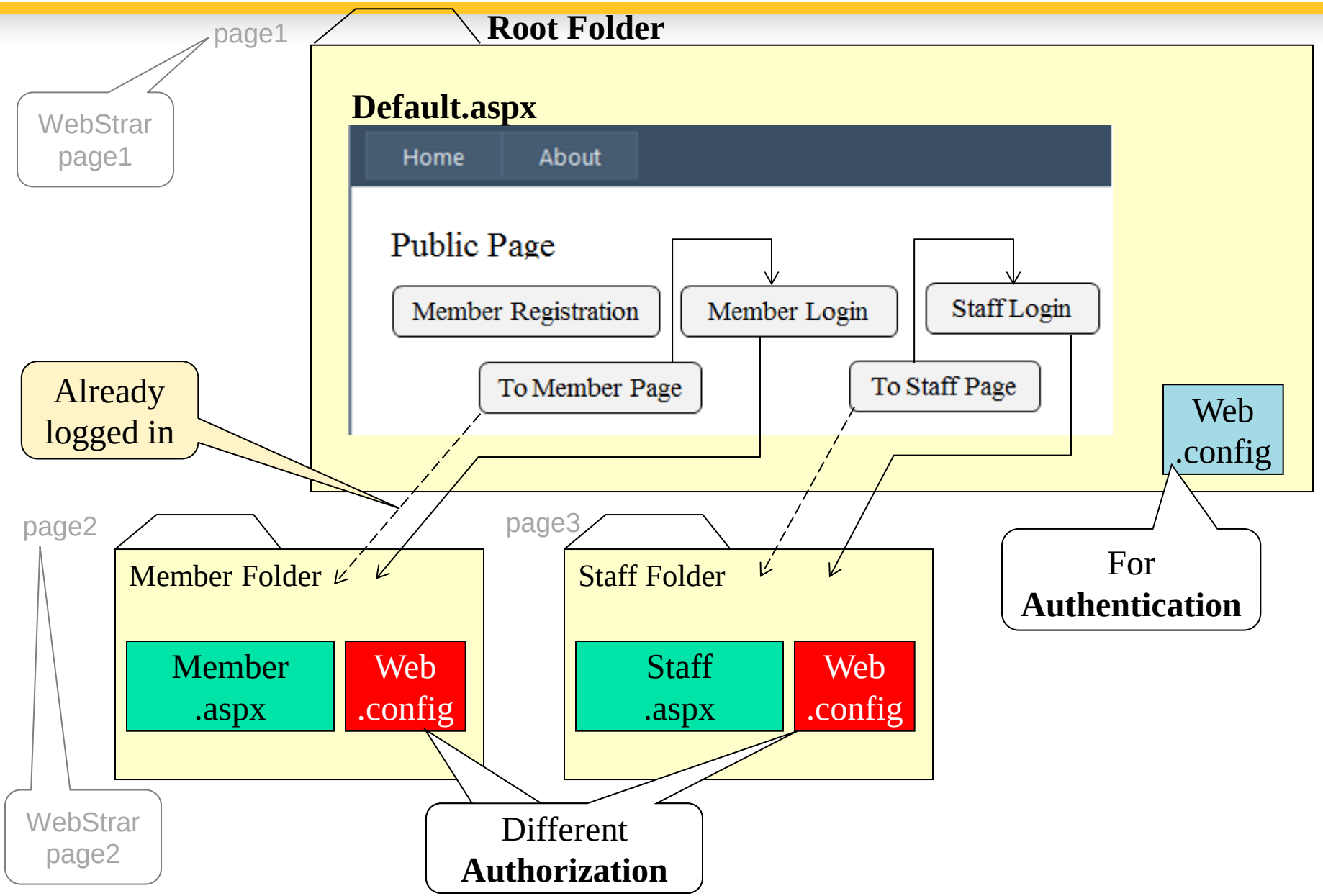
- The application's user interface consists of two content pages:
  1. Default.aspx, which can be viewed by anyone
  2. Staff.aspx, which is available only to authenticated users.
- The management pages consist of
  3. Login.aspx, which asks for a username and a password.
  4. The valid usernames and passwords stored in a Web.config in a virtual root directory
  5. Another Web.config in the sub directory
  6. And more…

Application Architectures

http://venus.sod.asu.edu/WSRepository/FormsSecurity/

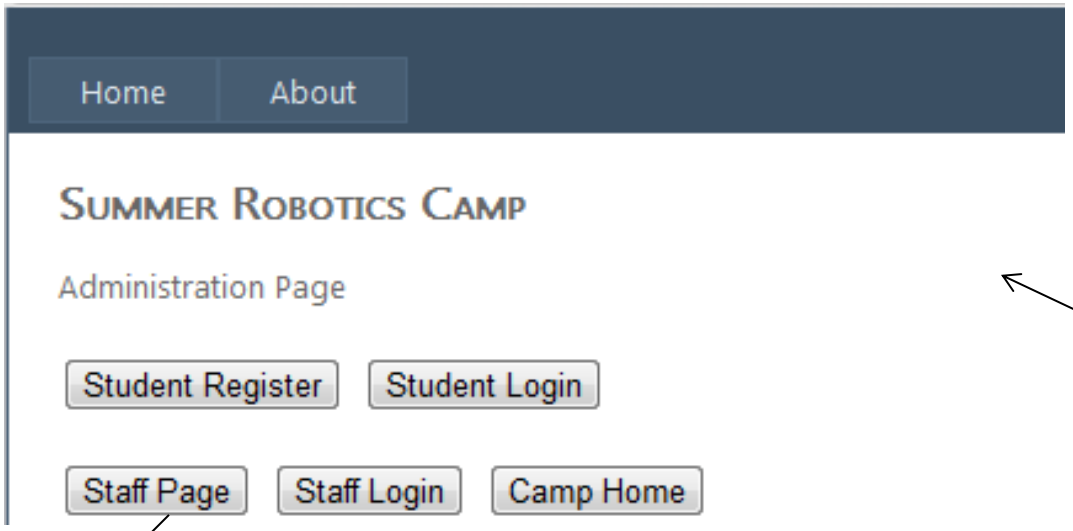


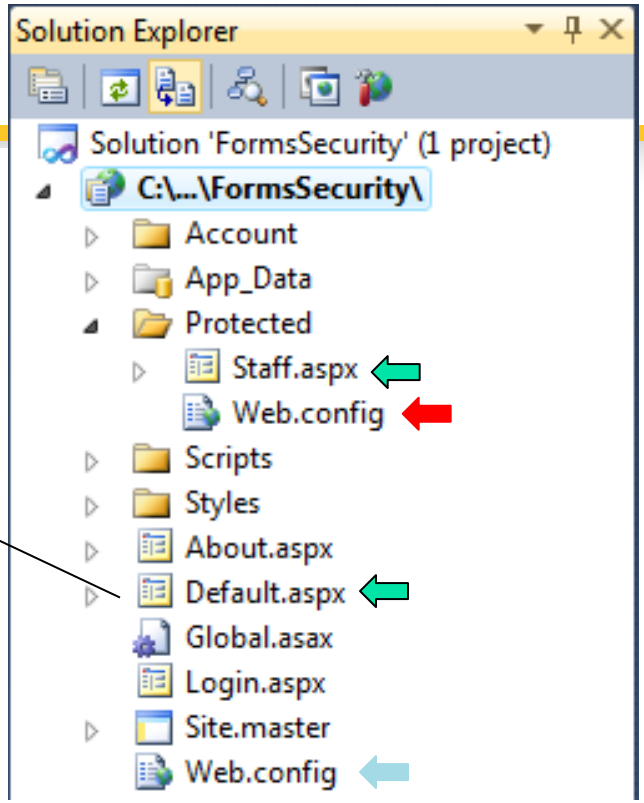
Course Project Organization Example



Default.aspx

http://venus.sod.asu.edu/WSRepository/FormsSecurity/






```
protected void btnStaff_Click(object sender, EventArgs e)
    {
        Response.Redirect("Protected/Staff.aspx");
    }
}
```

Transparently redirected to the Login.aspx, whose URL is stored in Web.config, then return to this page

<authentication> Element in Root Web.config

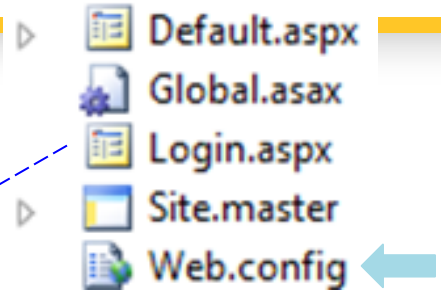
```
<authentication mode="Windows|Forms|Passport|None">
    <forms name="name"
            loginUrl="url_of_Login.aspx"
            protection="All|None|Encryption|Validation"
            timeout="30" path="/" >
            requireSSL="true|false"
            slidingExpiration="true|false">
      <credentials passwordFormat="Clear|SHA1|MD5">
          <user name="username" password="password"/>
      </credentials>
    </forms>
</authentication>
```

You could store usernames and passwords here, but it is not recommended.

Cryptographic Algorithms

- **SHA-1** (Secure Hash Algorithm, 0, 1, 2) is a cryptographic hash function designed by the National Security Agency and published by the NIST as a U.S. Federal Information Processing Standard.
- **MD5** (Message-Digest algorithm 5) is a widely used cryptographic hash function with a 128-bit (16-byte) hash value.
- There are many other cryptographic algorithms:
  - Asymmetric-key
  - Barrett reduction
  - DES (Data Encryption Standard) is an algorithm we will discuss later.

Web.config in the Application Root Directory




```
<configuration>
  <system.web>
 <authentication mode="Forms">
      <forms name = "LoginForm" loginUrl="Login.aspx" timeout="30" >
       <credentials passwordFormat="Clear">
          <username="Julia" password="abc" />
          <username="John" password="45678" />
          <username="Bob" password="123" />
          <username="Alice" password="12345" />
       </credentials>
      </forms>
    </authentication>
  </system.web>
</configuration>
```

Entry address stored here

→ SHA1 or MD5

Even if Clear is used, one **cannot** use "View Source" to read the file

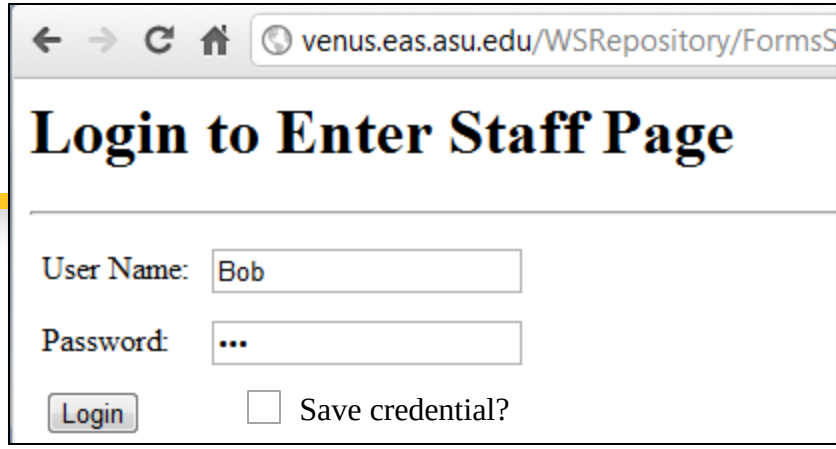
You could store usernames and passwords here, but it is not recommended.

Web.config in the “Protected” Sub Directory


```
<configuration>
    <system.web>
        <authorization>
            <deny users="?" />
        </authorization>
    </system.web>
</configuration>
```

?: Deny unauthenticated users

Login.aspx




```
<html>
 <body>
  <h1> Login to access Staff page
 </h1> <hr>
  <form runat="server">
   <table cellpadding="4">
    <tr>
     <td> User Name:  </td>
     <td>
      <asp:TextBox ID= "txtUserName"
 RunAt="server" />
     </td>
    </tr>
    <tr><td> Password: </td><td>
     <asp:TextBox ID= "txtPassword"
 TextMode="password"
      RunAt="server" />
     </td>
    </tr>
     <tr>
      <td>
        <asp:Button Text= "btnLogin"
 OnClick="LoginFunc" RunAt="server" />
      </td>
     </tr>
    </table>
</form>
  <hr>
  <h3><asp:Label ID="Output"  RunAt="server"/> </h3>
 </body>
</html>

<script language="C#" runat="server">
 void LoginFunc(Object sender, EventArgs e)
 {
    if (FormsAuthentication.Authenticate
              (txtUserName.Text,  txtPassword.Text))
       FormsAuthentication.RedirectFromLogin
              (UserName.Text, false);
    else
       Output.Text = "Invalid login";
 }
</script>
```

Look up the Web.config

Persistent option not checked.

Staff.aspx in the Protected Sub Directory


```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server"> <title>staff page</title> </head>
<body>
    <form id="form1" runat="server">
        <h1>Staff Page of the Camp</h1>
        <div>
        <% Response.Write("Hello " + Context.User.Identity.Name + ", "); %> <br />
        This page contains the information about staff members who will teach
        and manage the camp. Only authenticated users can access this page .<br />
        </div>
    </form>
</body>
</html>
```


Ira A. Fulton Schools of
Engineering
Arizona State University

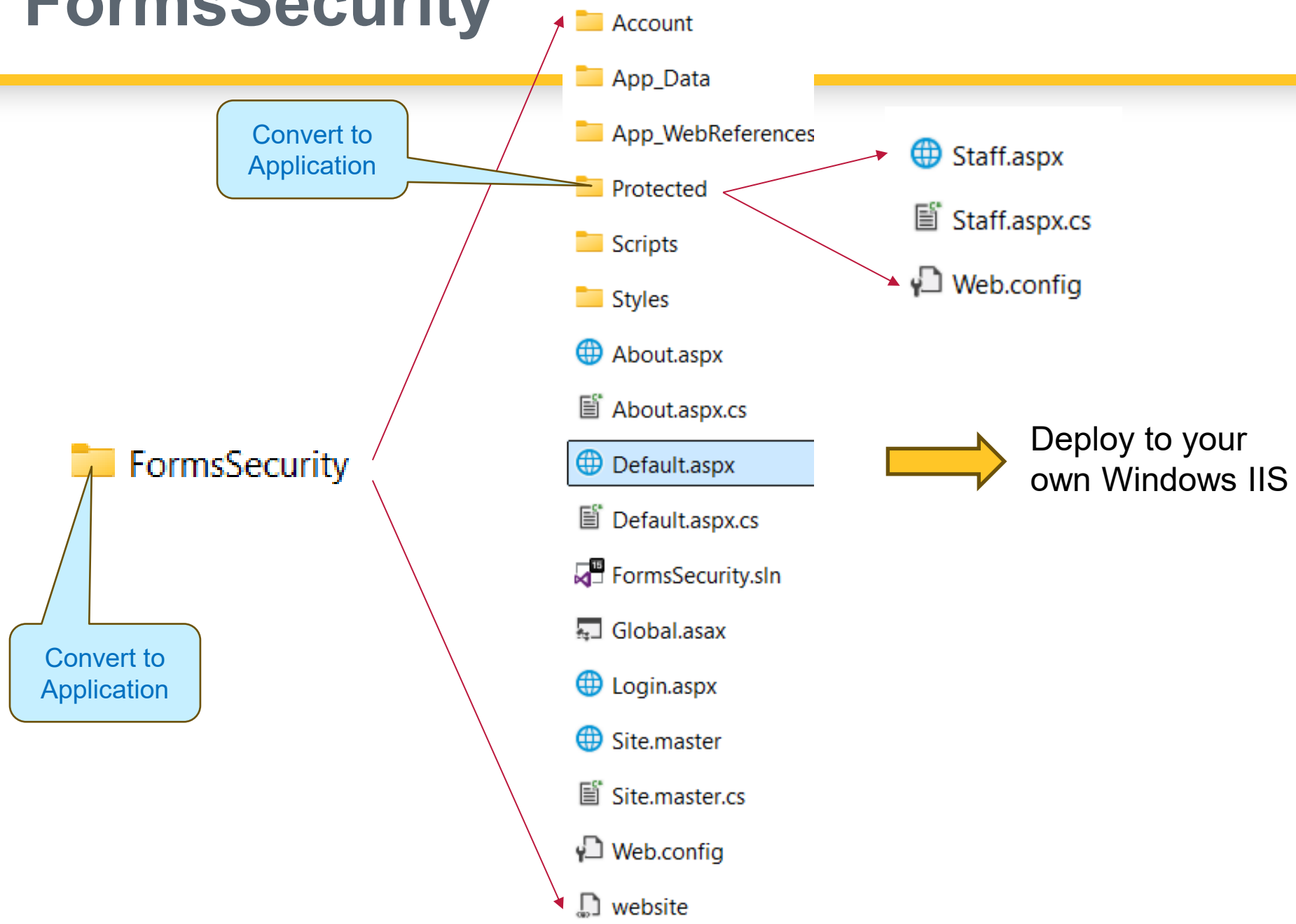
M13 L5
Forms Security Deployment and Testing

Ira A. Fulton Schools of Engineering
Arizona State University

Lecture Outline

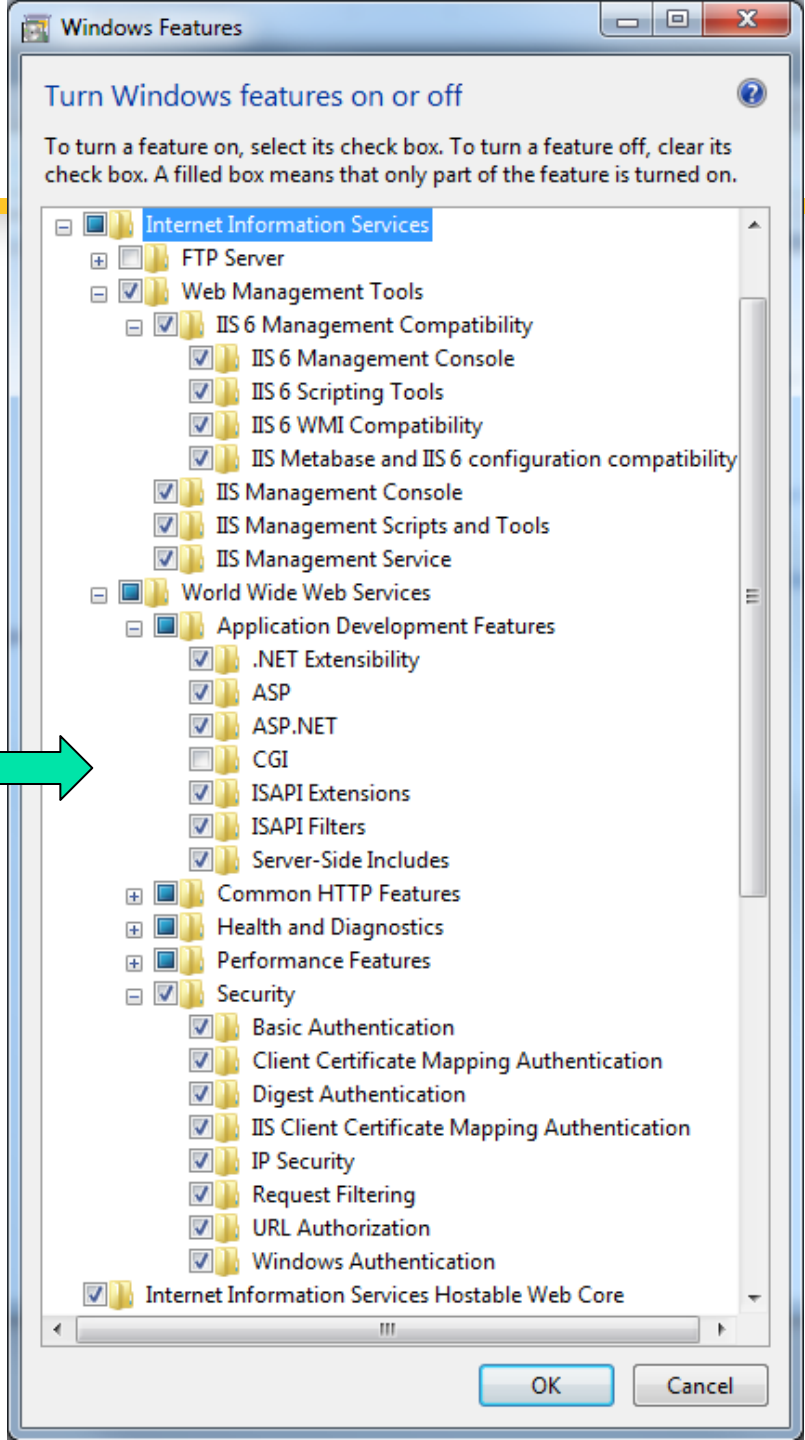
- Define authentication and Authorization
- Write My Own Authentication Method
- Saving Credential in XML File or Database

FormsSecurity



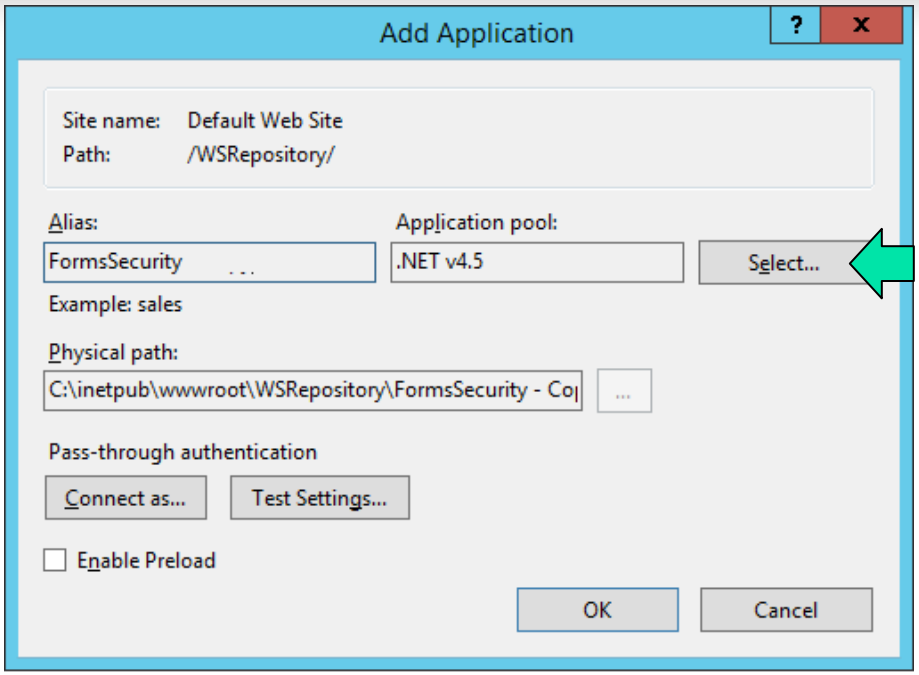
Deployment Example

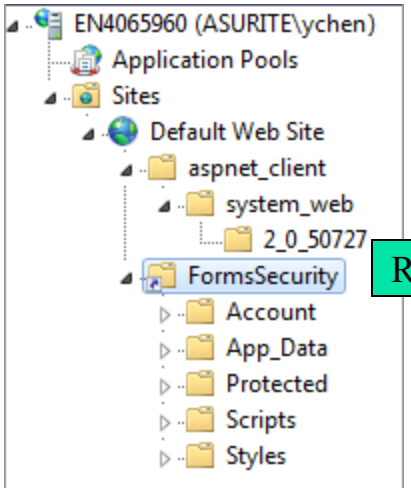
- You can use
  - .Net localhost / IIS Express
  - Your Windows IIS
  - Other Web / Cloud server
- If you use your own IIS or other self-managed server, you need to turn on your IIS:
  Control Panel → Programs → Programs and Features → Turn Windows Features on or off



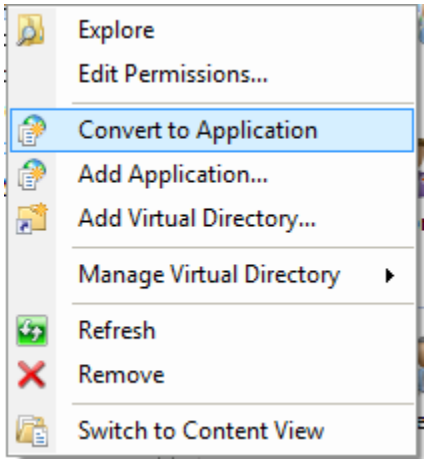
Create Virtual Directory and Convert to Application

- Administrative Tools → IIS Manager
- Create a virtual directory and link your project folder to the virtual directory
- Convert to Application

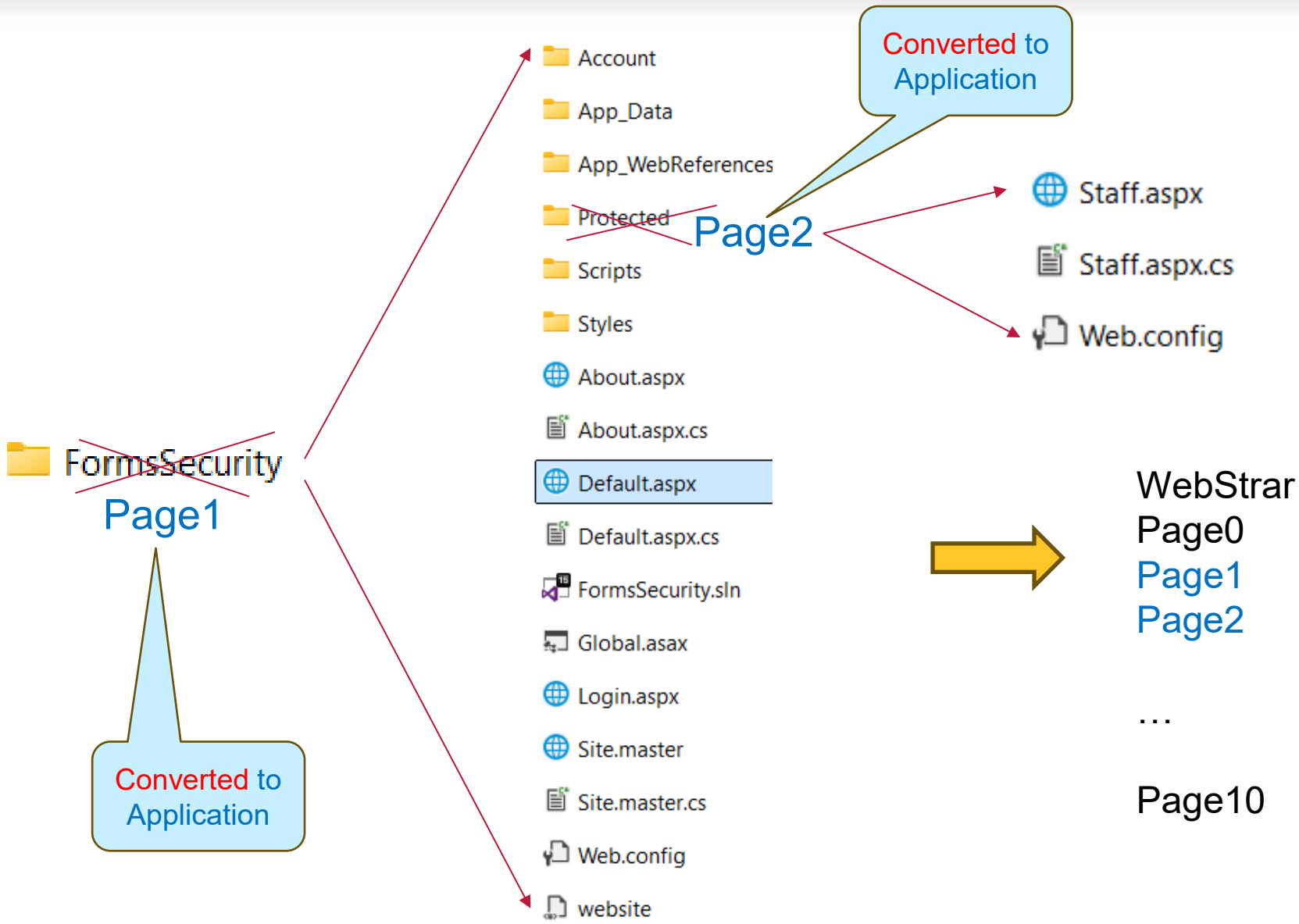




Right click

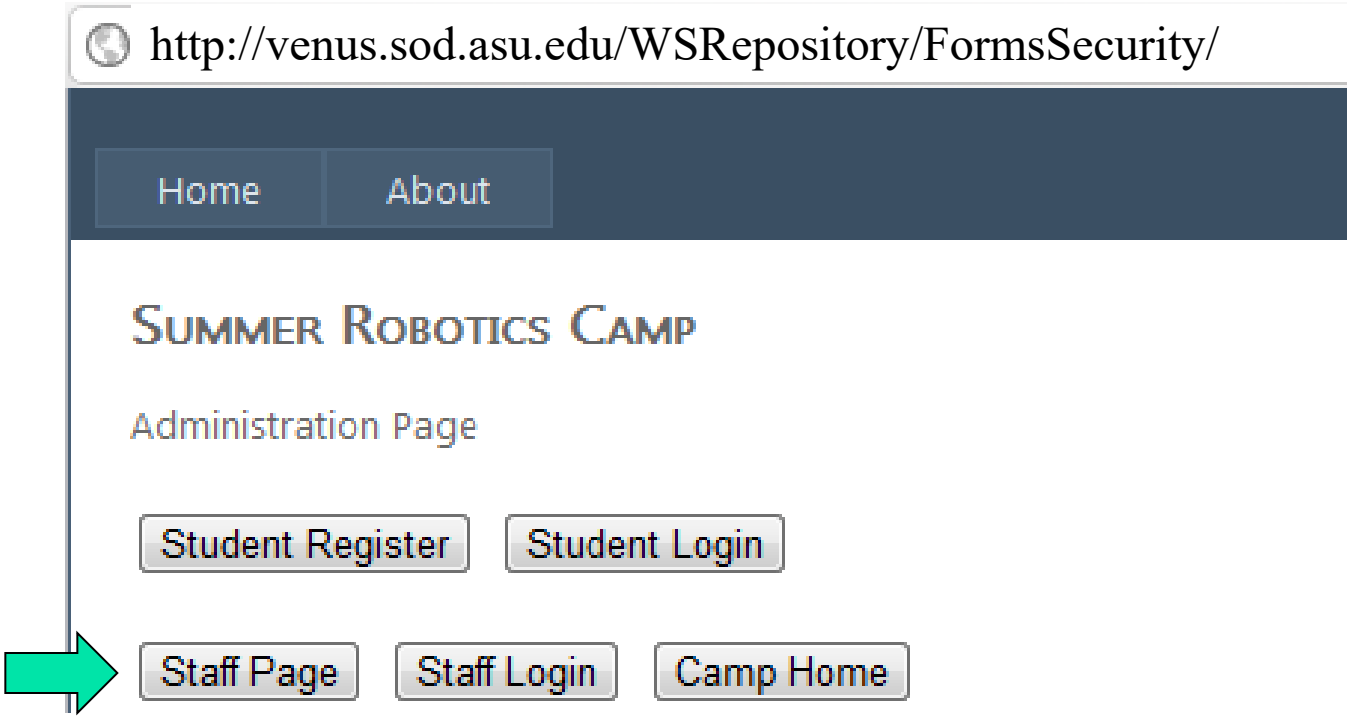


FormsSecurity Deploy to WebStrar



Testing in Action: Steps 1 and 2

1. Start the application

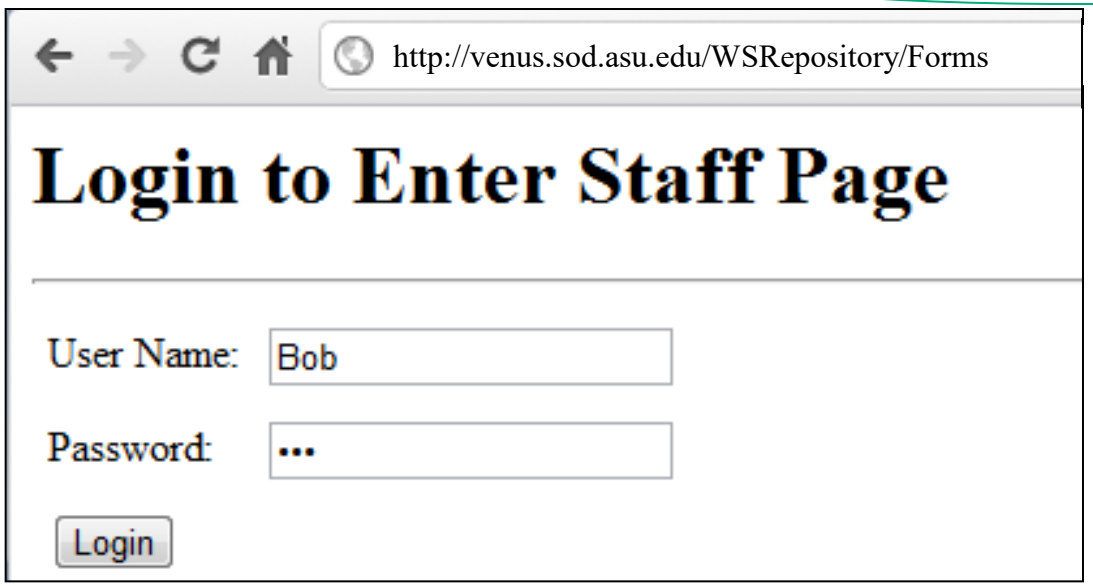


2. Click on "Staff" button

Steps 3 and 4 in Action

3. Staff page uses *Response.Redirect* to go to Protected/Staff.aspx. But because Staff.aspx is viewable only by authenticated users, ASP.NET displays the login form in Login.aspx.

http://venus.sod.asu.edu/WSRepository/FormsSecurity/Login.aspx?ReturnUrl=%2fWSRepository%2fFormsSecurity%2fProtected%2fStaff.aspx



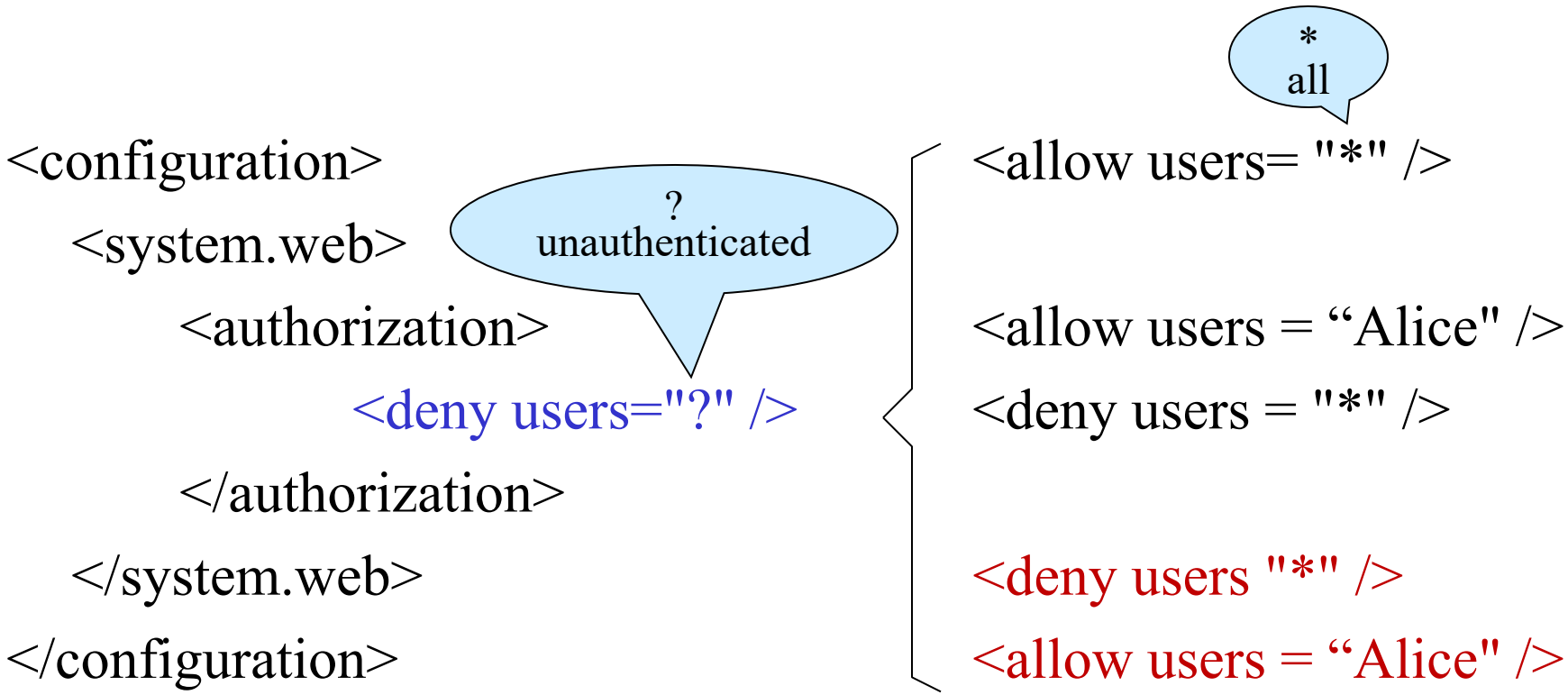
After login, redirect to Staff.aspx page

4. Type “Bob” into the User Name field and “123” into the Password field.

Step 5 in Action

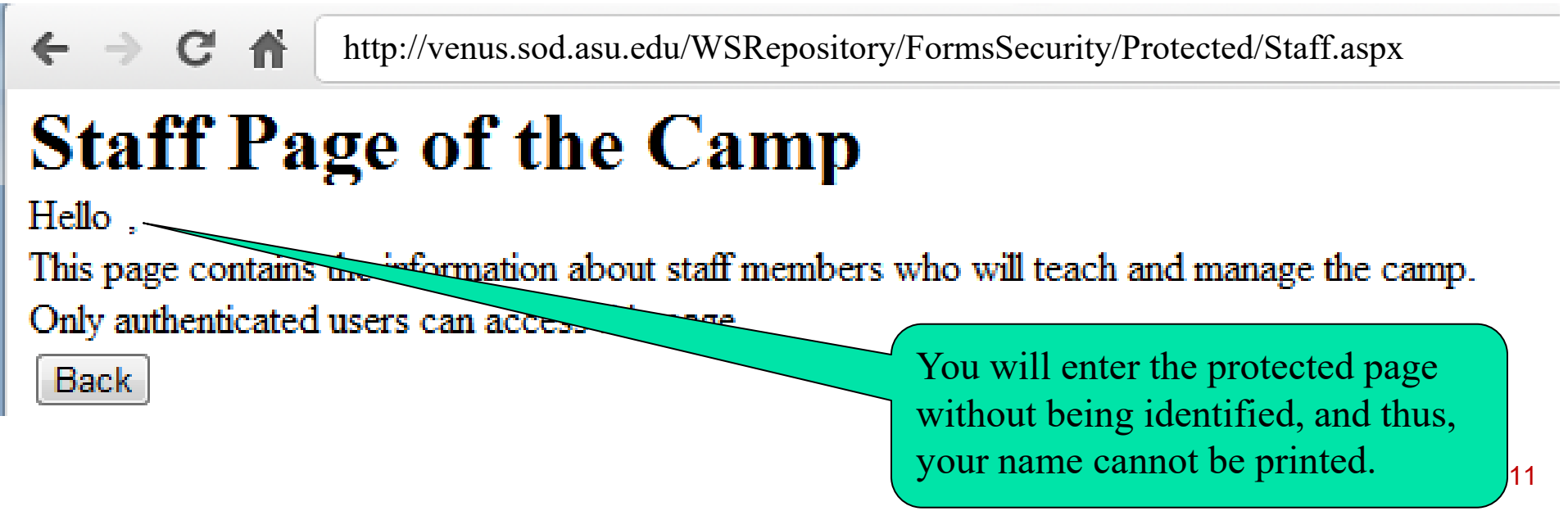
5. Staff.aspx appears. Because you are now an authenticated user, you have been issued an authentication ticket that accompanies subsequent requests as a cookie.

Configure the Authorization using the Web.config in the Protected Sub Directory



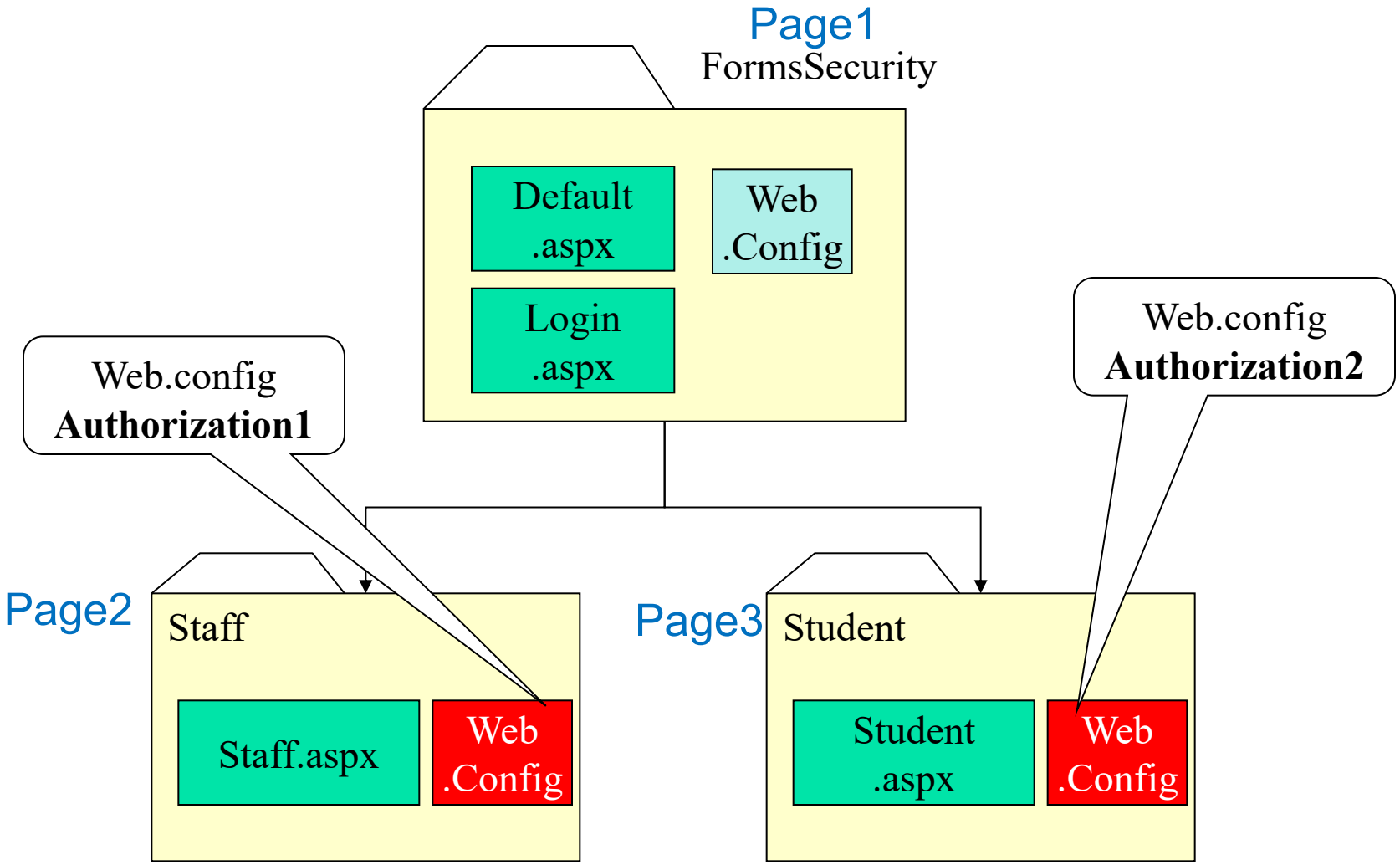
Apply <allow users="*" />

```
<configuration> <system.web>
        <authorization>
                <allow users="*" />
        </authorization>
    </system.web> </configuration>
```



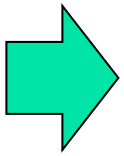
Different Pages with Different Authorizations

- How Do You Create Different Pages with Different Authorizations?



Problems of using Web.config for Authentication

- Users IDs and password are buried in many other data;
- Authentication is slow (sequential comparison) and unmanageable if the number of users is large;
- Authorization is unmanageable if accessibility needs to be changed from time to time.
- The method FormsAuthentication.Authenticate is obsolete: Not support in .Net 4.5 to discourage its use
- Force you to write your own authenticate methods to
  - Save credentials in XML file/XML database
  - Save credentials in relational databases
  - Save credentials in Web.config if you want to!



Write My Own Authentication Method (1)

- The application's interface consists of two pages:
  1. Default.aspx can be viewed by anyone
  2. Staff.aspx is available only to authenticated users.
- The management pages consist of
  3. Login.aspx, which asks for a username and a password.
  4. Web.config in the virtual root directory
  5. The valid usernames and passwords stored in a Web.config in a secret sub directory

Login.aspx: Part 1 HTML (GUI)

```
<html>
  <body>
    <h1>Please Log In</h1> <hr>
    <form runat="server">
        <table cellpadding="8">
            <tr>    <td>User Name:</td>
                    <td> <asp:TextBox ID="UserName" RunAt="server" /></td> </tr>
            <tr>    <td> Password: </td>
                    <td><asp:TextBox ID="Password" TextMode="password"
                                RunAt="server" /> </td> </tr>
            <tr>
                <td><asp:Button Text="Log In" OnClick="LoginFunc"
                        RunAt="server" /></td>
                <td><asp:CheckBox Text="Keep me signed in" ID="Persistent"
                        RunAt="server"/> </td>
            </tr>
        </table>
    </form>
    <hr> <h3><asp:Label ID="Output" RunAt="server" /></h3>
  </body>
</html>
```

continued

Login.aspx: Part 2 C# with My Own Method

```
<script language="C#" runat="server">
  void LoginFunc (Object sender, EventArgs e)
  {
     if (myAuthenticate (UserName.Text, Password.Text))
        FormsAuthentication.RedirectFromLogin
                    (UserName.Text, Persistent.Checked);
     else
        Output.Text = "Invalid login";
  }
```

It will automatically search Web.config for credentials

```
if (FormsAuthentication.Authenticate
                    (UserName.Text,  Password.Text))
```

continued

Login.aspx: Part 3, Using an XML File

```
bool myAuthenticate (string username, string password) {
    string fLocation = Path.Combine(Request.PhysicalApplicationPath,
    @"App_Data\Users.xml");
      if (File.Exists(fLocation))
      {
          FileStream FS= new FileStream(fLocation, FileMode.Open);
          XmlDocument xd = new XmlDocument();
          xd.Load(FS);
          XmlNode node = xd;
          XmlNodeList children = node.ChildNodes;
        foreach (XmlNode child in children)
        {
            // use hash function if the credential is hashed
            // check if the username and password exist in the XML file;
        }
    }
}
```

Login.aspx: Part 3, Accessing Database (Text Chapter 10)

```
bool myAuthenticate (string username, string password) {
    SqlConnection connection = new SqlConnection
                    ("server=localhost;database=weblogin;uid=sa;pwd=letme");
    try {
        connection.Open ();
        StringBuilder builder = new StringBuilder ();
        builder.Append ("select count (*) from users " + "where username = \' ");
        builder.Append (username);
        builder.Append ("\' and cast (rtrim (password) as " + "varbinary) = cast (\' ");
        builder.Append (password);
        builder.Append ("\' as varbinary)");
        SqlCommand command = new SqlCommand (builder.ToString (),
    connection);
        int count = (int) command.ExecuteScalar ();
        return (count > 0);
    }
    catch (SqlException) { return false; }
    finally { connection.Close (); }
 }
</script>
```

If count > 0, an entry is found!

We can make password more secure by not allowing "read" the database.

Signing Out

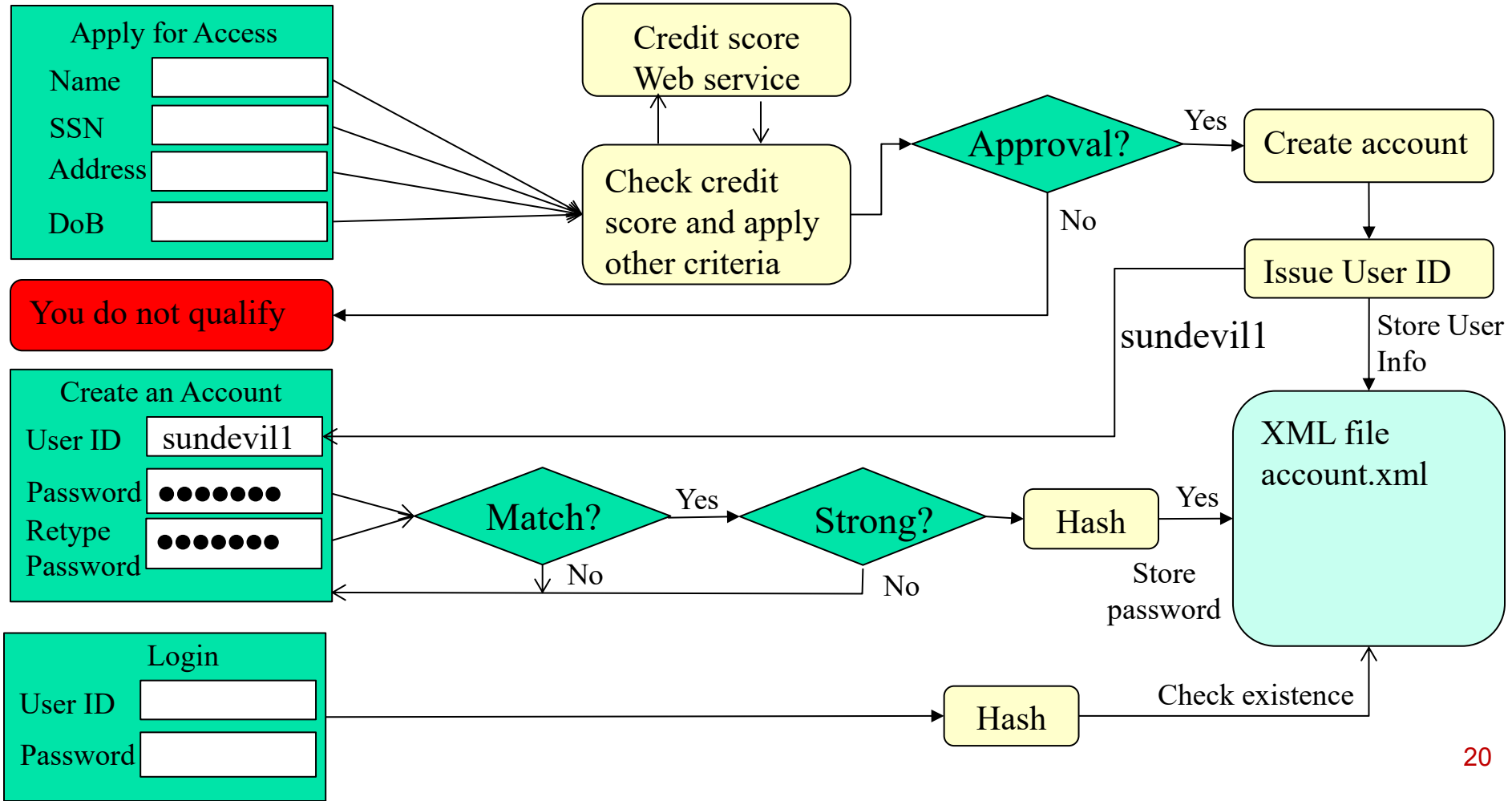
- It is always a good practice to provide the users a signing out button, for two reasons:
  - More secure
  - Resources can be recycled immediately

```
Void Signout(object sender, EventArgs e)
{
        FormsAuthentication.SignOut();
        Server.Transfer("Default.aspx");
}
```

Example: Applying for a Credit Account

- Apply for Access
- Check Credential

- Use ID to Create Account
- Issue an ID / Username




Ira A. Fulton Schools of
Engineering
Arizona State University

M13 L6
Forms Security: Self Registration and Cookie Support

Ira A. Fulton Schools of Engineering
Arizona State University

Lecture Outline

- User Registration & Account Management
- Solution and Project Stack
- Code Inspection

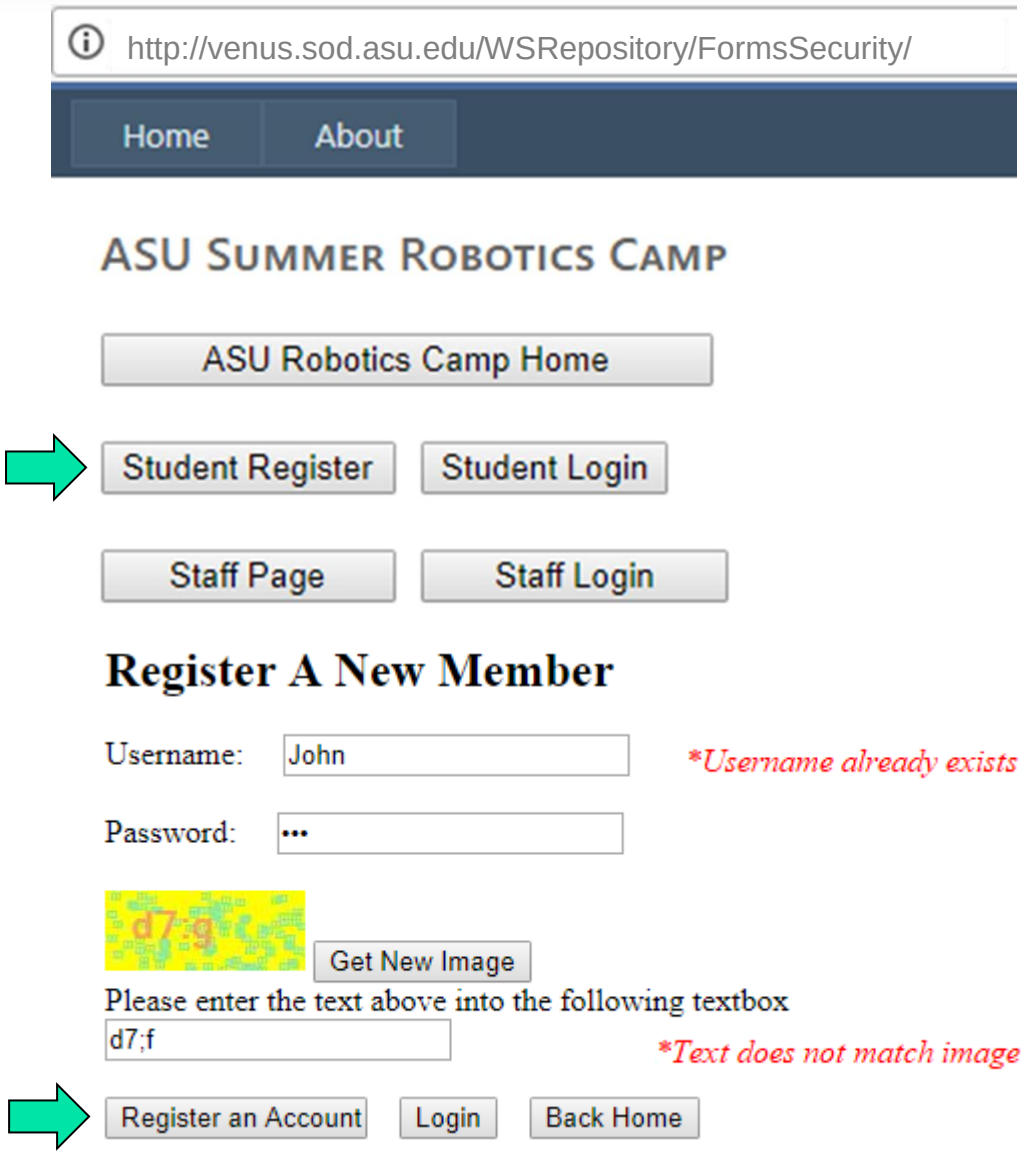
- Cookie Support to Forms Security

User Registration & Account Management

- So far, we have discussed how to authenticate and authorize users if they have already their credentials stored on the server of the application.
- We assumed that the administrator has created the credentials for the users.
- What should we do if we want to allow users to register and create their own credentials?
- ASP .Net has created standard Web controls to implement such functions.
- Let us continue with our Forms Security example…

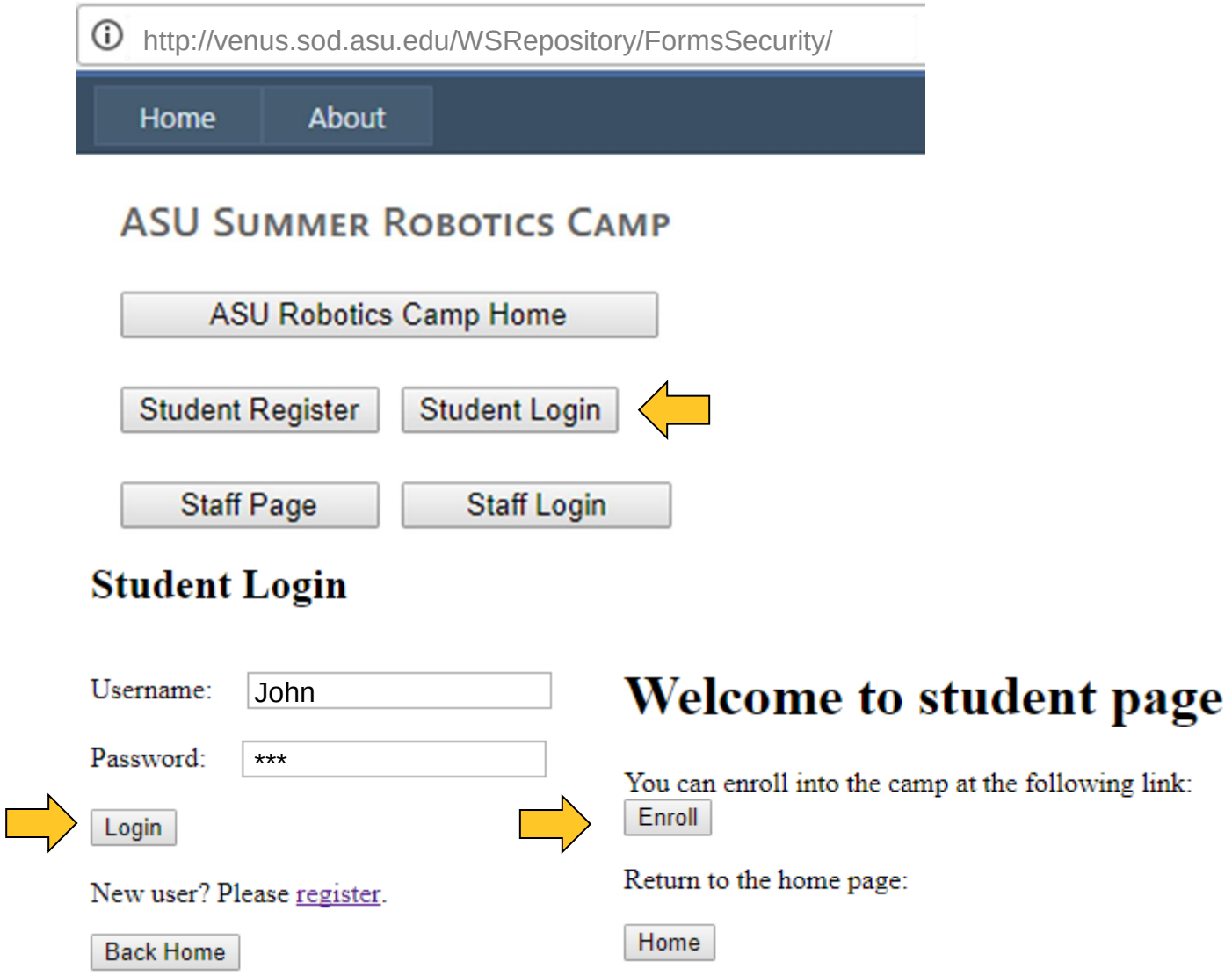
Case Study

http://venus.sod.asu.edu/WSRepository/FormsSecurity/

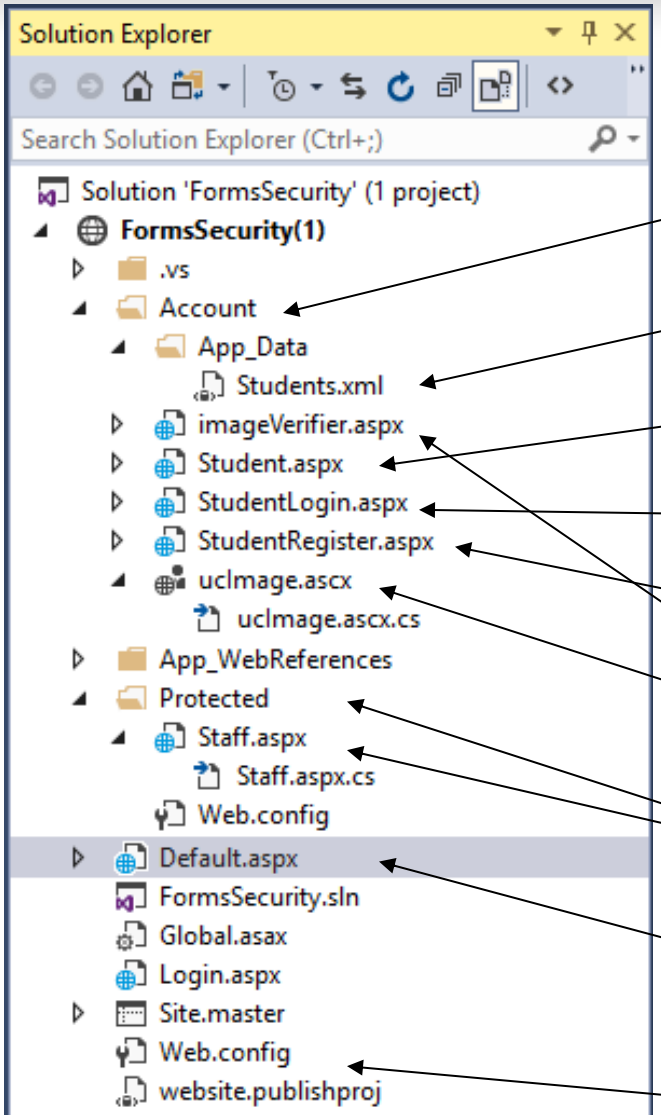


Case Study

http://venus.sod.asu.edu/WSRepository/FormsSecurity/



Solution and Project Stack



Self-enrollment Account folder

Store student credentials in XML

Or in a DB

Student information page

Student Login page

Student Register page

Image verifier user control

Staff page, protected by Login

Home page Default.aspx

Staff Login page, to protect Staff page

Code Behind Default.aspx Page

```
public partial class _Default : System.Web.UI.Page {
    protected void btnCamp_Click(object sender, EventArgs e) {
        Response.Redirect("http://venus.sod.asu.edu/roboticscamp/");
    }
    protected void btnStaff_Click(object sender, EventArgs e) {
        Response.Redirect("Protected/Staff.aspx"); // staff page
    }   // Note, from staff folder to parent: use ../
    protected void btnStaffLogin_Click(object sender, EventArgs e) {
        Response.Redirect("Login.aspx"); // Staff login page
    }
    protected void btnRegister_Click(object sender, EventArgs e) {
        Response.Redirect("Account/StudentRegister.aspx");
    }
    protected void btnStudentLogin_Click(object sender, EventArgs e) {
        Response.Redirect("Account/StudentLogin.aspx");
    }
}
```

Code Behind StudentRegister.aspx Page

HashRef: http://venus.sod.asu.edu/WSRepository/Services/HashSha512/Service.svc

```
string filepath = HttpRuntime.AppDomainAppPath+@"\Account\App_Data\Students.xml";
string user = txtUser.Text;
string password = txtPassword.Text;
HashRef.ServiceClient h = new HashRef.ServiceClient(); // hashing
string pwdEncrypt = h.Hash(password, "CSE445");
XmlDocument myDoc = new XmlDocument();
myDoc.Load(filepath);
                                                    // open file
XmlElement rootElement = myDoc.DocumentElement;
foreach (XmlNode node in rootElement.ChildNodes) {
    if (node["name"].InnerText == user) {
        errorUser.Text = String.Format("*Account with username {0}
already exists.", user);
        errorUser.Visible = true;
        return;
    }
}
errorUser.Visible = false;
```

Check duplicate

Register Page: Add a New User into XML

```
XmlElement myMember = myDoc.CreateElement("member",
        rootElement.NamespaceURI);
rootElement.AppendChild(myMember);
XmlElement myUser = myDoc.CreateElement("name",
        rootElement.NamespaceURI);
myMember.AppendChild(myUser);
myUser.InnerText = user;

XmlElement myPwd = myDoc.CreateElement("pwd",
        rootElement.NamespaceURI);
myMember.AppendChild(myPwd);
myPwd.InnerText = pwdEncrypt;

myDoc.Save(filepath);
```

Add username

Add password

The Code Behind the StudentLogin.aspx

http://venus.sod.asu.edu/WSRepository/Services/HashSha512/Service.svc

```
string filepath = HttpRuntime.AppDomainAppPath+@"\Account\App_Data\Students.xml";
string user = txtUser.Text; string password = txtPassword.Text;
HashRef.ServiceClient h = new HashRef.ServiceClient(); // hashing
string pwdEncrypt = h.Hash(password, "CSE445");
XmlDocument myDoc = new XmlDocument();
myDoc.Load(filepath);                                    // open file
XmlElement rootElement = myDoc.DocumentElement;
foreach (XmlNode node in rootElement.ChildNodes) {
    if (node["name"].InnerText == user){
        if (node["pwd"].InnerText == pwdEncrypt) {
            errorLogin.Visible = false;
            createLoginCookie();
            Response.Redirect("Student.aspx");
            return;
        }
        else {//if username exists but password does match.
            errorLogin.Visible = true;
            return;
}   }   }
errorLogin.Visible = true; return;
```

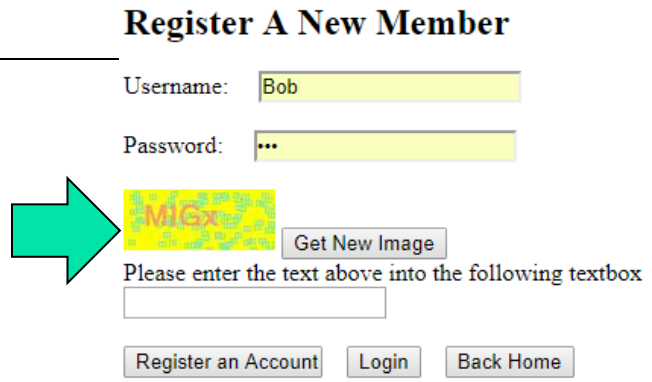
Check existence

User Control for image verification

ucImage.ascx

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="ucImage.ascx.cs" Inherits="Account_ucImage" %>

<asp:Image ID ="ImageString" runat="server"> </asp:Image>
```



ucImage.ascx.cs

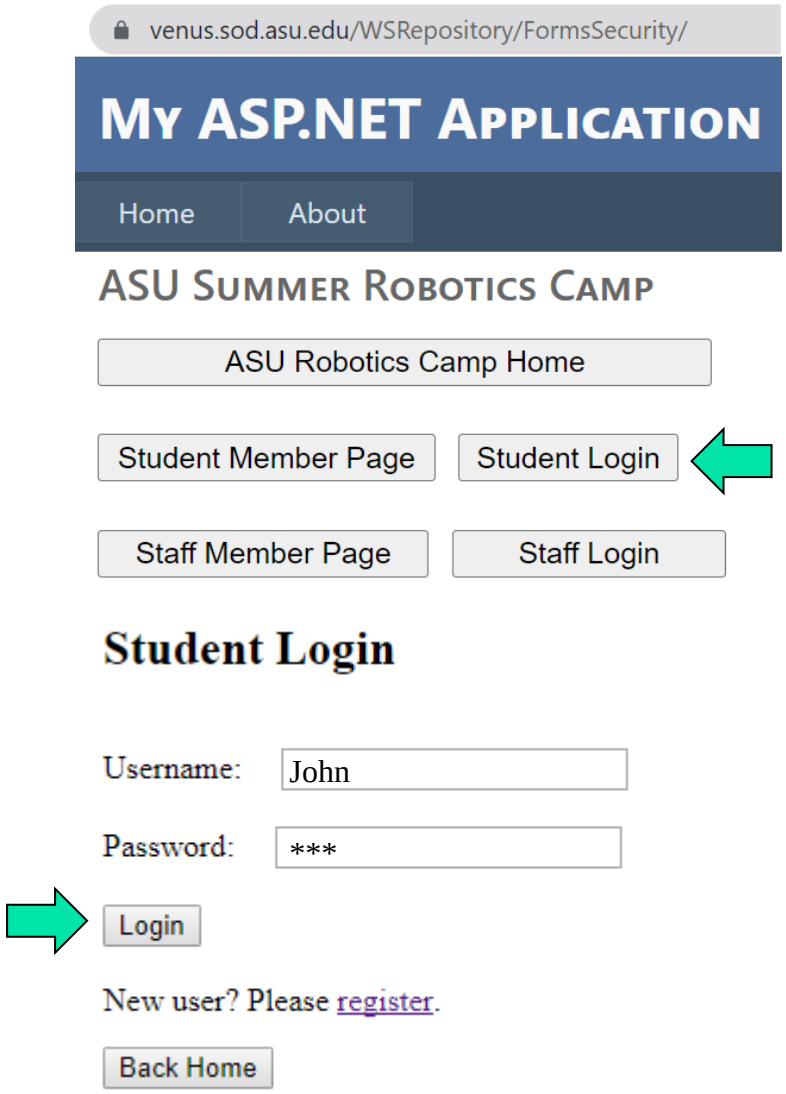
```
public partial class Account_ucImage :
System.Web.UI.UserControl
{
    protected void Page_Load(object sender, EventArgs e)
    {
        //Use image string created in imageVerifier.aspx
        ImageString.ImageUrl = "imageVerifier.aspx?";
    }
}
```

imageVerifier.aspx

- Review previous lecture using the image verifier service
- Read the textbook Section 7.3.4 for implementation detail of an Image Verifier.

Demonstration

http://venus.sod.asu.edu/WSRepository/FormsSecurity/



Cookie Support to Forms Security

Ira A. Fulton Schools of Engineering
Arizona State University

Cookie Support to Forms Security

- Role of Cookies in Forms Security
- Web.config Setting
- Options in Forms-Based Security

The Roles of Cookies in Forms Security

- Client's credential is stored on server side and can also be stored in client machine's hard drive as cookies;
- When the client revisits the Web application, the cookies are sent to the server and compared with the credential stored on server-side to authenticate the client.
- Cookies are secure. They are of string type and are encrypted:
  - Open key encryption can be applied: both confidential and digital signature (validation) can be applied.
  - SSL can be required for transmitting the credential.

Cookieless Setting in Forms Security

- Cookies are used in Session state for the similar purpose:
  - Where, 120-bit session id is issued and sent to client
  - Developers can choose to UseCookies or cookieless (UseUri)
- Cookieless is supported for both Session State and Forms Security. If UseUrl is selected, the credential will be store in the browser, and URL will be used for transmitting the credential:
  - Absolute path for URL cannot be used in programming;
  - When relative URL is used, a system method will be used to create the full URL, which can then perform credential encoding and decoding
  - In this case, the credential will be saved while the browser is still running. If browser closed, the credential is discarded.

Cookie in Forms Section in Web.config (Pattern)

```
<forms
    name = "CookieName"
    loginUri = "URL"
    defaultUrl="URL"
    protection = "All | Encryption | Validation | None"
    Timeout = "10"
    requireSSL = "true | false"
    slidingExpiration="true | false"
    path = "/"
    enableCrossApplicationRedirects = "true | false"
    cookieless = "UseCookies | UseUri | AutoDetect | UserDeviceProfile"
    domain = "domain name"
</forms>
```

Default: Both encryption & validation will be applied to cookies store on client side.

Cookies are signed (digital signature), so that people cannot modify the cookies. Once modified, they will fail validation.

Cookies are encrypted, so that people cannot read the cookies.

Cookie Attributes in <forms> Section in Web.config

Attribute	Description
cookieless	Four different values, also see previous chapter
defaultUrl	Defines the default URL that is used for redirection after authentication.
domain	Specifies an optional domain to set on outgoing forms-authentication cookies. This setting takes precedence over the domain that is used in the httpCookies element.
enableCross AppRedirects	This attribute can be one of the following values. **True** Specifies that authenticated users can be redirected to URLs in other Web applications. **False** Specifies that authenticated users cannot be redirected to URLs in other Web applications.
loginUrl	Specifies the URL to which the request is redirected for logon, if no valid authentication cookie is found, the default is login.aspx.
name	Optional attribute. Specifies the HTTP cookie to use for authentication. If multiple applications are running on a single server and each application requires a unique cookie, you must configure the cookie name in each Web.config file for each application.
path	Specifies the path for cookies that are issued by the application.

Cookie Attributes in <forms> Section in Web.config (contd.)

Attribute	Description
protection (four possible values)	**All** (Default) Use both data ***validation*** and ***encryption*** to protect the cookie. **Encryption**: the cookie is encrypted by using open key system 3DES or DES. Cookies used in this manner might be subject to chosen plain-text attacks. **Validation**: use a digital signature to verify that the contents of an encrypted cookie have not been changed in transit. The cookie is created by concatenating a validation key with the cookie data, computing a Message Authentication Code (MAC), and appending the MAC to the outgoing cookie. **None** Specifies that both encryption and validation are disabled for sites that are using cookies only for personalization and that have less stringent security requirements.
requireSSL	**True** Specifies that an SSL connection is required to help protect the user's credentials during transmission. **False** (Default) Specifies that an SSL connection is not required.
sliding Expiration	**True** (Default) Sliding expiration is enabled. The authentication cookie is refreshed and the time to expiration is reset on subsequent requests during a single session. **False** The cookie expires at a set interval from the time the cookie was originally issued.
timeout	Specifies the time in minutes

Saving Encrypted Password into Cookies

http://msdn.microsoft.com/en-us/library/System.Web.Security.FormsAuthentication(v=vs.110).aspx

In the .Net framework class library, there are functions like

- FormsAuthentication.Encrypt Method: Creates a string containing an encrypted forms-authentication ticket suitable for use in an HTTP cookie.

```
public static string Encrypt(
    FormsAuthenticationTicket ticket )
```

- **FormsAuthentication.Decrypt Method:** Creates a FormsAuthenticationTicket object based on the encrypted forms-authentication ticket passed to the method.

```
public static FormsAuthenticationTicket Decrypt(
    string encryptedTicket )
```

Summary of Cookies Applications

Three applications are discussed

1. Explicitly use cookies as storage to store any client information:
   - Explicitly read and write Cookie[“key”]
   - The cookies are not automatically encrypted;
2. Implicitly used in Session state to identify the revisiting browser and to access session storage.
   - When a Session state is created, a session ID is generated and send to the client machine and stored in cookies
   - Session ID is automatically encrypted (hashed)
3. Implicitly used in Forms security:
   - It can automatically store the credentials into cookies.
   - Credentials are encrypted in Default setting.


Ira A. Fulton Schools of
Engineering
Arizona State University