



TestKingprep.com

**MCSE, CCNA, CCNP, OCP, CIW, JAVA, Sun Solaris, Checkpoint
World No 1 Cert Exams**

70-340

Congratulations!

You have purchased a TestKingprep.com Study Guide.

This study guide is a complete collection of questions and answers that have been developed by our professional & certified team. You must study the contents of this guide properly in order to prepare for the actual certification test. The average time that we would suggest you for studying this study guide is approximately 10 to 20 hours and you will surely pass your exam. We guarantee it!

GOOD LUCK!

DISCLAIMER

This study guide and/or material is not sponsored by, endorsed by or affiliated with Microsoft, Cisco, Oracle, Citrix, CIW, Checkpoint, Novell, Sun/Solaris, CWNA, LPI, ISC, etc. All trademarks are properties of their respective owners.

Guarantee

If you use this study guide correctly and still fail the exam, send a scanned copy of your official score notice at: Sales@TestKingprep.com

We will gladly refund the cost of this study guide or give you an exchange of study guide of your choice of the Free.

This material is protected by copyright law and international treaties. Unauthorized reproduction or distribution of this material, or any portion thereof, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under law.

Total No of Questions:-64

QUESTION NO: 1

You are an application developer for Testkingprep.in. You develop library assemblies that are called by your main applications. These library assemblies access confidential data in the applications. To ensure that this data is not accessed in an unauthorized and unsafe manner, users must not be allowed to call the library assemblies from their own applications. You apply a strong name to all assemblies to support versioning.

You need to prevent users from writing managed applications that make calls to your library assemblies.

You need to achieve this goal while minimizing the impact on response times for applications.

What should you do?

- A. Use the internal access modifier to declare all classes and structures in each library.
- B. Use the protected internal access modifier to declare all classes and structures in each library.
- C. Add the following attribute to each class and structure in each library assembly:
<StrongNameIdentityPermission(SecurityAction.Demand,
PublicKey:="002400..bda4")>
- D. Add the following attribute to each class and structure in each library assembly:
<StrongNameIdentityPermission(SecurityAction.LinkDemand,
PublicKey:="002400..bda4")>

Answer: C

Explanation:

StrongNameIdentityPermission Class

Defines the identity permission for strong names. This class cannot be inherited.

For a list of all members of this type, see StrongNameIdentityPermission Members.

System.Object

System.Security.CodeAccessPermission

System.Security.Permissions.StrongNameIdentityPermission

<Serializable>

NotInheritable Public Class StrongNameIdentityPermission

Inherits CodeAccessPermission

Remarks

Use StrongNameIdentityPermission to achieve versioning and naming protection by confirming that the calling code is in a particular strong-named code assembly.

A strong name identity is based on a cryptographic public key called a blob optionally combined with the name

and version of a specific assembly. The key defines a unique namespace and provides strong verification that

the name is genuine, because the definition of the name must be in an assembly signed by the corresponding

private key.

Note that the validity of the strong name key is not dependent on a trust relationship or any certificate

necessarily being issued for the key.

Note **Full demands for StrongNameIdentityPermission succeed only if all the assemblies in the stack**

have the correct evidence to satisfy the demand. Link demands using

StrongNameIdentityPermissionAttribute succeed if only the immediate caller has the correct evidence.

Demands

You can use the security demand call declaratively or imperatively to specify the permissions that direct or

indirect callers must have to access your library. Direct callers explicitly call static or instance methods of your

library, while indirect callers call static or instance methods of another library that calls your library. **When you**

use a demand, any application that includes your code will execute only if all direct and indirect callers

have the permissions that the demand specifies. Demands are particularly useful in situations in which your

class library uses protected resources that you do not want to be accessed by untrusted code. Demands can be

placed in code using either imperative or declarative syntax.

Note that most classes in the .NET Framework already have demands associated with them, so you do not need

to make an additional demand whenever you use a class that accesses a protected resource.

Link Demands

A link demand causes a security check during just-in-time compilation and checks only the immediate caller of

your code. Linking occurs when your code is bound to a type reference, including function pointer references

and method calls. If the caller does not have sufficient permission to link to your code, the link is not allowed

and a runtime exception is thrown when the code is loaded and run. **Link demands can be overridden in**

classes that inherit from your code.

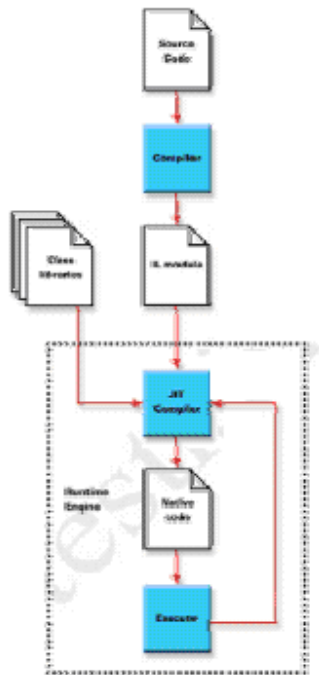
Just-in-Time compilation

Languages in the .NET Framework compile to Microsoft Intermediate Language (IL) ready for the JIT (Just-in-

Time) compiler to turn them into native code when the program is installed or first run.

The runtime engine

pulls in uncompiled functions for compilation on the fly as required.



The following example shows how to demand that the calling code has **StrongNameIdentityPermission** at **link time**. Code will only execute if signed with a strong name using the private key counterpart of the specified public key

```

<StrongNameIdentityPermissionAttribute(SecurityAction.Demand, _
PublicKey := "00240000048000009400000006020000002400005253413100040000010"
& _
"00100538a4a19382e9429cf516dcf1399facdcca092a06442efaf9ecaca33457be26ee0"
& _
"073c6bde51fe0873666a62459581669b510ae1e84bef6bcb1aff7957237279d8b7e0e25b"
& _
"71ad39df36845b7db60382c8eb73f289823578d33c09e48d0d2f90ed4541e1438008142e"
& _
"f714bfe604c41a4957a4f6e6ab36b9715ec57625904c6")> Public Class SampleClass

```

Restrict Unauthorized Code

By using .NET Framework code access security — specifically, code identity demands — you can limit the assemblies that can access your data access classes and methods. For example, if you only want code written by your company or a specific development organization to be able to use your data access components, use a **StrongNameIdentityPermission** and demand that calling assemblies have a strong name with a specified public key, as shown in the following code fragment: using **System.Security.Permissions**;

...

```
[StrongNameIdentityPermission(SecurityAction.LinkDemand,  
PublicKey="002...4c6")]  
public void GetCustomerInfo(int CustId)  
{ }
```

To extract a text representation of the public key for a given assembly, use the following command:

```
sn -Tp assembly.dll
```

Note Use an uppercase "T" in the -Tp switch.

Because Web application assemblies are dynamically compiled, you cannot use strong names for these

assemblies. This makes it difficult to restrict the use of a data access assembly to a specific Web application.

The best approach is to develop a custom permission and demand that permission from the data access

component. Full trust Web applications (or any fully trusted code) can call your component. Partial trust code,

however, can call your data access component only if it has been granted the custom permission.

When You Should Use Demands?

Your code is always subject to permission demand checks from the .NET Framework class library, but if your

code uses explicit permission demands, check that this is done appropriately. Search your code for the

".Demand" string to identify declarative and imperative permission demands, and then review the following questions:

- **Do you cache data?**

If so, check whether or not the code issues an appropriate permission demand prior to accessing the cached

data. For example, if the data is obtained from a file, and you want to ensure that the calling code is

authorized to access the file from where you populated the cache, demand a

FileIOPermission prior to

accessing the cached data.

- **Do you expose custom resources or privileged operations?**

If your code exposes a custom resource or privileged operation through unmanaged code, check that it

issues an appropriate permission demand, which might be a built-in permission type or a custom permission

type depending on the nature of the resource.

- **Do you demand soon enough?**

Check that you issue a permission demand prior to accessing the resource or performing the privileged

operation. Do not access the resource and then authorize the caller.

- **Do you issue redundant demands?**

Code that uses the .NET Framework class libraries is subject to permission demands.

Your code does not

need to issue the same demand. This results in a duplicated and wasteful stack walk.

When to Use Link Demands?

Link demands, unlike regular demands, only check the immediate caller. They do not perform a full stack walk,

and as a result, **code that uses link demands is subject to luring attacks**. For information on Luring Attacks,

see "Link Demands" in Chapter 8, "Code Access Security in Practice."

Search your code for the ".LinkDemand" string to identify where link demands are used.

They can only be used

declaratively. An example is shown in the following code fragment:

```
[StrongNameIdentityPermission(SecurityAction.LinkDemand,  
PublicKey="00240000048...97e85d098615")]
```

```
public static void SomeOperation() {}
```

- **Why are you using a link demand?**

A defensive approach is to avoid link demands as far as possible. Do not use them just to improve performance and to eliminate full stack walks. Compared to the costs of other Web application performance issues such as network latency and database access, the cost of the stack walk is small. Link demands are only safe if you know and can limit which code can call your code.

- **Do you trust your callers?**

When you use a link demand, you rely on the caller to prevent a luring attack. Link demands are safe only

if you know and can limit the exact set of direct callers into your code, and you can trust those callers to

authorize their callers.

- **Do you call code that is protected with link demands?**

If so, does your code provide authorization by demanding a security permission from the callers of your

code? Can the arguments passed to your methods pass through to the code that you call?

If so, can they

maliciously influence the code you call?

- **Have you used link demands at the method and class level?**

When you add link demands to a method, it overrides the link demand on the class.

Check that the

method also includes class-level link demands.

- **Do you use link demands on classes that are not sealed?**

Link demands are not inherited by derived types and are not used when an overridden method is called on

the derived type. If you override a method that needs to be protected with a link demand, apply the link

demand to the overridden method.

- **Do you use a link demand to protect a structure?**

Link demands do not prevent the construction of a structure by an untrusted caller.

This is because

default constructors are not automatically generated for structures, and therefore the structure level link

demand only applies if you use an explicit constructor.

- **Do you use explicit interfaces?**

Search for the **Interface** keyword to find out. If so, check if the method implementations are marked with

link demands. If they are, check that the interface definitions contain the same link demands. Otherwise, it

is possible for a caller to bypass the link demand.

The allowed access modifiers in .NET are public, private, protected, internal, and protected internal.

These keywords control the visibility of class members (and other things), defining the circumstances under

which a member may be accessed—hence their collective name as access modifiers.

With the exception of the

last, protected internal, it's illegal to combine two access modifiers.

Public means just that: public and visible to everyone and everything. A public member can be accessed using

an instance of a class, by a class's internal code, and by any descendants of a class.

Private is also intuitively understood, meaning hidden and usable only by the class itself.

No code using a class

instance can successfully access a private member and neither can a descendant class.

Protected members are similar to private ones in that they are accessible only by the containing class.

However, protected members also may be used by a descendant class. So members that are likely to be needed

by a descendant class should be marked protected.

Members marked as internal are public to the entire application but private to any outside applications.

Internal is useful when you want to allow a class to be used by other applications but reserve special

functionality for the application that contains the class.

Finally, we have **the only compound access modifier allowed in .NET, protected internal**. Members marked

as protected internal may be accessed only by a descendant class that's contained in the same application as its

base class. You use protected internal in situations where you want to deny access to parts of a class'

functionality to any descendant classes found in other applications.

Not limited to controlling class access

As mentioned before, access modifiers aren't limited to use on class members but can be applied to a few othercode constructs. The rules defining when modifiers may be legally assigned to a construct are dependant on the construct's container:

- *Interface* and *enumeration* members are always public and no access modifiers are needed (or allowed).
- Classes in *namespaces* are internal by default and may be either internal or public, while *namespaces* themselves are always public.
- Members of a *struct* are private by default and may be given public, internal, or private

access modifiers.

QUESTION NO: 2

You are an application developer for Testkingprep.in. You are developing an application that can be extended by using custom components. The application uses reflection to dynamically load and invoke these custom components. In some cases, custom components will originate from a source that is not fully trusted, such as the Internet.

You need to programmatically restrict the code access security policy under which custom components run so that custom components do not run with an elevated permission grant.

What are two possible ways to achieve this goal? (Each correct answer presents a complete solution. Choose two)

- A. Create a new application domain and set the security policy level. Run custom components in this application domain.
- B. Use permission class operations to modify the security policy.
- C. Implement custom permission classes to protect custom component resources.
- D. Programmatically modify the machine-level security policy file after loading a custom component.

Answer: B, C

Explanation:

All permission objects must implement the `IPermission` interface. Inheriting from the `CodeAccessPermission` class is the easiest way to create a custom permission because **CodeAccessPermission** implements

IPermission and provides most of the methods required for a permission. Additionally, you must implement the

`IUnrestrictedPermission` interface for all custom code access permissions. The custom permission class is

required for both imperative and declarative security support, so you should create it even if you plan to use

only declarative security.

Defining the Permission Class

To derive from the **CodeAccessPermission** class, you must override the following five key methods and

provide your own implementation:

- **Copy** creates a duplicate of the current permission object.
- **Intersect** returns the intersection of allowed permissions of the current class and a passed class.
- **IsSubsetOf** returns **true** if a passed permission includes everything allowed by the current permission.
- **FromXml** decodes an XML representation of your custom permission.
- **ToXml** encodes an XML representation of your custom permission.

The **IUnrestrictedPermission** interface requires you to override and implement a single method called

IsUnrestrictedPermission. In order to support the **IUnrestrictedPermission** interface, you must implement some system, such as a Boolean value that represents the state of restriction in the current object, to define whether the current instance of the permission is unrestricted.

Note Custom permissions should either be marked as sealed (NotInheritable in Visual Basic) or have an inheritance demand placed on them. Otherwise, malicious callers are able to derive from your permission, potentially causing security vulnerabilities.

Default security policy does not know about the existence of any custom permission. For example, the

Everything named permission set contains all the built-in code access permissions that the runtime provides, but it does not include any custom permissions. To update security policy so that it knows about your custom permission, you must do three things:

- Make policy aware of your custom permission.
- Add the assembly to the list of trusted assemblies.
- Tell security policy what code should be granted to your custom permission.

Making Policy Aware of Your Custom Permission

To make policy aware of your custom permission, you must:

- Create a new named-permission set that includes your custom permission. (You can modify an existing named-permission set instead of creating a new one.)
- Give the permission set a name.
- Tell security policy that the named-permission set exists.

For more information, see Code Access Security Policy Tool (Caspol.exe) or .NET Framework Configuration

Tool (Mscorcfg.msc). You can add a new permission set in one of several ways. Using the Code Access

Security Policy tool (Caspol.exe), you can create an .xml file that contains an XML representation of a custom

permission set and then add this file to the security policy on the computer where the code is to run. Using the

.NET Framework Configuration tool (Mscorcfg.msc), you can copy an existing permission set and add an XML representation of a permission to the new permission set.

To guarantee that your XML representation is valid and correctly represents your permission, you can generate

it using code similar to the example that follows. Notice that this code creates a custom permission called

MyCustomPermission, initialized to the unrestricted state. If your custom permission does not implement

`UnrestrictedPermission`, or if you do not want to set policy to grant your permission in an unrestricted state, use

the constructor to initialize your permission to the state you want it to have.

Application domains provide a more secure and versatile unit of processing than the common language runtime

can use to provide isolation between applications. Application domains are typically created by runtime hosts, which are responsible for bootstrapping the common language runtime before an application is run.

You can run several application domains in a single process with the same level of isolation that would exist in separate processes, but without incurring the additional overhead of making cross-process calls or switching between processes. The ability to run multiple applications within a single process dramatically increases server scalability. Isolating applications is also important for application security. For example, you can run controls from several

Web applications in a single browser process in such a way that the controls cannot access each other's data and resources.

The isolation provided by application domains has the following benefits:

- Faults in one application cannot affect other applications. Because type-safe code cannot cause memory faults, using application domains ensures that code running in one domain cannot affect other applications

in the process.

- Individual applications can be stopped without stopping the entire process. Using application domains enables you to unload the code running in a single application.

Note You cannot unload individual assemblies or types. Only a complete domain can be unloaded.

- Code running in one application cannot directly access code or resources from another application. The common language runtime enforces this isolation by preventing direct calls between objects in different application domains. Objects that pass between domains are either copied or accessed by proxy. If the object is copied, the call to the object is local. That is, both the caller and the object being referenced are in the same application domain. If the object is accessed through a proxy, the call to the object is remote. In this case, the caller and the object being referenced are in different application domains. Cross-domain calls use the same remote call infrastructure as calls between two processes or between two machines. As such, the metadata for the object being referenced must be available to both application domains to allow the method call to be JIT-compiled properly. If the calling domain does not have access to the metadata for the

object being called, the compilation might fail with an exception of type

System.IO.FileNotFound. See

Accessing Objects in Other Application Domains Using .NET Remoting for more details. The mechanism

for determining how objects can be accessed across domains is determined by the object. For more

information, see `MarshalByRefObject` Class.

- The behavior of code is scoped by the application in which it runs. In other words, the application domain provides configuration settings such as application version policies,

the location of any remote assemblies it accesses, and information about where to locate assemblies that are loaded into the domain.

- Permissions granted to code can be controlled by the application domain in which the code is running.

QUESTION NO: 3

You are an application developer for your company, which is named Testkingprep.in. You are developing an ASP.NET Web application that users in the accounting department will use to process payroll reports and view payroll reports. The application will use Integrated Windows authentication to authenticate all users.

Because payroll data is confidential only users in the accounting department will be granted access to the application. All employees in the accounting department belong to a specific Active Directory group. However, users in the IT department can add themselves to various Active Directory groups in order to troubleshoot resource access problems. These IT department users must not be granted access to the ASP.NET Web application.

The following rules can be used to distinguish between users in the accounting department and users in the IT department:

- **All users in the accounting department are members of a group named Testkingprep\Accounting.**
- **Some users in the IT department are members of the Testkingprep\Accounting group.**
- **All users in the IT department are members of a group named Testkingprep\Domain Admin.**
- **No users in the accounting department are members of the Testkingprep\Domain Admin group.**

You need to configure URL authorization for the application by adding an <authorization> element to the Web.config file in the application root.

Which element should you use?

- A. <authorization>
<deny roles="Testkingprep\Domain Admin"/>
<allow roles="Testkingprep\Accounting"/>
<deny users="*/>
</authorization>
- B. <authorization>
<allow roles="Testkingprep\Accounting"/>
<deny roles="Testkingprep\Domain Admin"/>
<deny users="*/>
</authorization>
- C. <authorization>
<deny roles="Domain Admin"/>
<allow roles="Accounting"/>
<deny users="*/>
</authorization>

```
D. <authorization>
<allow roles="Accounting"/>
<deny roles="Domain Admin"/>
<deny users="?" />
</authorization>
```

Answer: A

Explanation

<authorization> Element

Configures ASP.NET authorization support. The **<authorization>** tag controls client access to URL resources.

This element can be declared at any level (machine, site, application, subdirectory, or page).

```
<configuration>
<system.web>
<authorization>
<authorization>
<allow users="comma-separated list of users"
roles="comma-separated list of roles"
verbs="comma-separated list of verbs"/>
<deny users="comma-separated list of users"
roles="comma-separated list of roles"
verbs="comma-separated list of verbs"/>
</authorization>
```

Subtag Description

<allow> Allows access to a resource based on the following:

users: A comma-separated list of user names that are granted access to the resource. A question

mark (?) allows anonymous users; an asterisk (*) allows all users.

roles: A comma-separated list of roles that are granted access to the resource.

verbs: A comma-separated list of HTTP transmission methods that are granted access to the

resource. Verbs registered to ASP.NET are GET, HEAD, POST, and DEBUG.

<deny> Denies access to a resource based on the following:

users: A comma-separated list of user names that are denied access to the resource. A question

mark (?) indicates that anonymous users are denied access; an asterisk (*) indicates that all users

are denied access.

roles: A comma-separated list of roles that are denied access to the resource.

verbs: A comma-separated list of HTTP transmission methods that are denied access to the

resource. Verbs registered to ASP.NET are GET, HEAD, POST, and DEBUG.

Remarks

At run time, the authorization module iterates through the **<allow>** and **<deny>** tags until it finds the first access rule that fits a particular user. It then grants or denies access to a URL resource depending on whether the first access rule found is an **<allow>** or a **<deny>** rule. The default authorization rule in the Machine.config file is **<allow users="*" />** so, by default, access is allowed unless configured otherwise.

Example

The following example allows access to all members of the Admins role and denies access to all users.

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

QUESTION NO: 4

You are an application developer for Testkingprep.in. You develop an ASP.NET Web application for Testkingprep's intranet. The application accesses data that is stored in a Microsoft SQL Server database. The application authenticates users by using Windows authentication, and it has impersonation enabled. You configure database object permissions based on the identity of the user of the application. You need to provide the user's identity to the SQL Server database. What should you do?

- A. Connect to the database by using the following connection string
"Persist Security Info=False;Integrated Security=SSPI;
database=ApplicationDB;server=DataServer;"
- B. Connect to the database by using the following connection string
"User ID=ASPNET;Persist Security Info=False;Integrated
Security=False;
database=ApplicationDB;server=DataServer;"
- C. Develop a serviced component that wraps all database operations.
Use COM+ role-based security to restrict access to database operations based on user identity.
- D. Disable impersonation.

Answer: A

We need to configure the following four different areas to access Windows integrated security:

1. SQL Server
2. IIS Web Server
3. ASP.Net web application
4. ConnectionString

SQL Server be running on same IIS machine. If both are on different machines, we should go for an alternative security model such as Forms authentication, or Kerberos delegation would need to be used. The access users must be in the same domain where the Web server is running.

Configuring SQL Server

To configure SQL Server for Windows integrated security:

1. From the Windows **Start** menu, choose **Microsoft SQL Server**, and then choose **Enterprise Manager**.
2. Open the node for the server and expand the node for the database you want to give users permissions for.
3. Right-click the **Users** node and choose **New Database User**.
4. In the **Database User Properties** dialog box, enter *domain\username* in the **Login name** box, and then click **OK**. Alternatively, configure the SQL Server to allow all domain users to access the database.

Configuring IIS

You need to configure your application in IIS to turn off anonymous access and turn on Windows

authentication. To configure IIS for Windows integrated security:

1. In Windows, open the Internet Information Services administration tool.
2. Open the node for your server, and then open nodes until you find the node for your application, typically under Default Web Site.
3. Right-click your application and choose **Properties**.
4. In the **Directory Security** tab, click **Edit**.
5. In the **Authentication Methods** dialog box, clear the **Anonymous Access** box and make sure **Integrated Windows authentication** is checked.
6. Click **OK** to close all the dialog boxes.

Configuring the ASP.NET Web Application

In the application configuration file (Web.config), you establish the authentication mode that your application

uses and establish that the application will impersonate the user's credentials—that is, that it will run as that

user. To configure Web.config to allow Windows integrated security:

Open the Web.config file for your application and add the following elements to it:

```
<authentication mode="Windows" />
```

```
<identity impersonate="true"/>
```

The <authentication> element might already be there.

Creating Connection Strings

When you create a connection string to access SQL Server, you must include attributes that tell SQL Server that

you are using Windows integrated security. To configure connection strings for Windows integrated security:

In any connection string for SQL Server, include the Trusted_Connection=Yes attribute and remove the

username and password attributes. The following shows a typical connection string configured for Windows

integrated security:

```
"data source=Sql01;initial catalog=Northwind;  
integrated security=SSPI;persist security info=False;  
Trusted_Connection=Yes."
```

Sample C# code for connecting SQL server from ASP.Net application using windows authentication:

```
private void DataBind() {  
    sqlConnection = SqlConnection("data source=bondugula;  
    initial catalog=Northwind;  
    integrated security=SSPI;  
    persist security info=False;  
    Trusted_Connection=Yes");  
    sqlConnection.Open();  
    sqlDataAdapter = new SqlDataAdapter(  
    "SELECT EmployeeID, FirstName, LastName, Title  
    FROM Employees", sqlConnection);  
    dataSet = new DataSet();  
    sqlDataAdapter.Fill(dataSet, "Employees");  
    DataGrid1.DataSource = dataSet.Tables["Employees"].DefaultView;  
    DataGrid1.DataBind();  
}
```

Important settings in the web.config file are as follows:

```
<system.web>  
authentication mode = "Windows"/>  
<identity impersonate="true"/>  
<authorization>  
<allow users = "*" />  
</authorization>  
<!--other settings-->  
</system.web>
```

QUESTION NO: 5

You are an application developer for Testkingprep.in. You create an ASP.NET Web application that all authenticated network users will access. The authentication mode in the Web.config file is currently set to None. Due to recent security threats, the network administrator requires that all connections to the application's Web server use the network credentials of the authenticated user.

You need to configure the application to use the network credentials of the authenticated user as

HttpContext.Current.User.

Which action or actions should you perform? (Choose all that apply)

- A. Ask the network administrator to configure the IIS directory security to Anonymous authentication.
- B. Ask the network administrator to configure the IIS directory security to Integrated Windows authentication.
- C. Set the authentication mode in the Web.config file to **Forms**.
- D. Set the authentication mode in the Web.config file to **Windows**.
- E. Set the impersonation attribute of the identity element in the Web.config file to **true**.

Answer: B, D, E

Explanation

Authentication is the process of obtaining identification credentials such as name and password from a user and validating those credentials against some authority. If the credentials are valid, the entity that submitted the credentials is considered an authenticated identity. Once an identity has been authenticated, the authorization process determines whether that identity has access to a given resource.

Integrated Windows Authentication

Integrated Windows authentication uses Windows logon credentials to authenticate users. Rather than prompt

a user for a user name and password and transmit them over HTTP, a browser asked to identify the user through

integrated Windows authentication carries on a conversation with the Web server and identifies the user using

that person's login identity on the client.

ASP.NET implements authentication through authentication providers, the code modules that contain the code

necessary to authenticate the requestor's credentials. ASP.NET supports the authentication providers described in the following table.

ASP.NET authentication provider Description

Forms authentication A system by which unauthenticated requests are redirected to an HTML form using HTTP client-side redirection. The user provides credentials and submits the form. If the application authenticates the request, the system issues a cookie that contains the credentials or a key for reacquiring the identity. Subsequent requests are issued with the cookie in the request headers; they are authenticated and authorized by an ASP.NET event handler using whatever validation method the application developer specifies.

Passport authentication Centralized authentication service provided by Microsoft that offers a single logon and core profile services for member sites.

Windows authentication ASP.NET uses Windows authentication in conjunction

with Microsoft Internet Information Services (IIS) authentication. Authentication is performed by IIS in one of three ways: basic, digest, or Integrated Windows Authentication. When IIS authentication is complete, ASP.NET uses the authenticated identity to authorize access.

To enable an authentication provider for an ASP.NET application, you only need to create an entry for the application configuration file as follows.

// Web.config file

```
<authentication mode= "[Windows|Forms|Passport|None]"/>
```

The mode is set to one of the authentication modes: **Windows**, **Forms**, **Passport**, or **None**. The default is

Windows. If the mode is **None**, ASP.NET does not apply any additional authentication to the request - this can

be useful when you want to implement a custom authentication scheme, or if you are solely using anonymous

authentication and want the highest possible level of performance.

The authentication mode cannot be set at a level below the application root directory. As is the case with other

ASP.NET modules, subdirectories in the URL space inherit authentication modules unless explicitly overridden.

The **WindowsAuthenticationModule** provider relies on Microsoft Internet Information Services (IIS) to provide

authenticated users, using any of the mechanisms that IIS supports. If you want to implement site security with

a minimum of ASP.NET coding, this is the provider configuration you should use. The provider module

constructs a **WindowsIdentity** object. The default implementation constructs a

WindowsPrincipal object and

attaches it to the application context. The **WindowsPrincipal** object maps identities to Windows groups.

If you use IIS authentication, the provider module uses the authenticated identity passed in from IIS. IIS

authenticates the identity using basic, digest, or Integrated Windows authentication, or some combination of

them. You can use impersonation and NTFS ACL permissions to restrict or allow access to protected resources.

An important reason to use the **WindowsAuthenticationModule** provider is to implement an impersonation

scheme that can use any of the authentication methods that might have already been performed by IIS before

passing the request to the ASP.NET application. To do this, set the authentication mode to **Windows**, and

confirm that the **impersonate** element is set to **true**, as shown in the following example:

```
<authentication mode="Windows"/>
```

```
<identity impersonate="true"/>
```

Please note that configuring an ASP.NET application has no effect on the IIS Directory Security settings.

The systems are completely independent and are applied in sequence. In addition to selecting an authentication mode for an ASP.NET application, it is also important to configure IIS authentication appropriately.

Next, you must set the NTFS ACLs to allow access only to the proper identities. If you want to enable impersonation for only a short time during request processing, you can do it by using an impersonation context and **WindowsIdentity.Impersonate**.

First, set the **impersonate** element to **false**, and then set up a context using the **WindowsIdentity.Impersonate**

method, as shown in the following example.

```
Dim context As WindowsImpersonationContext = _  
WindowsIdentity.Impersonate(impersonateToken)  
' Perform some action.  
context.Undo()
```

Notice that you can use **context.Undo** for identity reversion.

As mentioned earlier, you can implement a custom Windows authorization scheme by using a

WindowsAuthentication_OnAuthenticate event handler to create a **WindowsPrincipal** or a **GenericPrincipal** object from a **WindowsIdentity** object. You can then use one of the new objects to implement your own custom authentication scheme. The **WindowsPrincipal** object maps identities to Windows groups. The default implementation constructs a **WindowsPrincipal** object and attaches it to the application context.

QUESTION NO: 6

You are an application developer for Testkingprep.in. You develop a Windows Forms application. You want your application to use a class library that was developed by another developer. You run the Permissions View tool on the class library and receive the following output.

Microsoft (R) .NET Framework Permission Request Viewer. Version
1.1.4322.573

Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.
minimal permission set:

```
<PermissionSet class="System.Security.PermissionSet"  
version="1">
```

```
<IPermission class="System.Security.Permissions.FileIOPermission,  
mscorlib,
```

```
Version=1.05000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"  
version="1"
```

```
Write="C:\SecureFile.txt"/>
```

```

<IPermission class="System.Security.Permissions.ReflectionPermission,
mscorlib,
Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
version="1"
Flags="ReflectionEmit"/>
IPermission class="System.Security.Permission.SecurityPermission,
mscorlib,
Version=1.0.5000.0, Culture=netrual, PublicKeyToken=b77a5c561934e089"
version="1"
Flags="SerializationFormatter"/>
</PermissionSet>
optional permission set:
<PermissionSet class="System.Security.PermissionSet"
version="1"
Unrestricted="true"/>
refused permission set:
Not specified

```

**You need to add the correct attributes to the Windows Forms application code before the call to the class library.
Which code segment should you use?**

- A. <Assembly: ReflectionPermission(SecurityAction.RequestMinimum, _
ReflectionEmit:=False), _
Assembly: SecurityPermission(SecurityAction.RequestOptional, _
SerializationFormatter:=False), _
Assembly: FileIOPermissionAttribute(SecurityAction.RequestRefuse, _
Write:="C:\SecureFile.txt"), _
Assembly: PermissionSetAttribute(SecurityAction.RequestOptinal, _
Unrestricted:=Trye)>
- B. <Assembly: ReflectionPermission(SecurityAction.RequestOptional, _
ReflectionEmit:True), _
Assembly: SecurityPermission(SecurityAction.RequestMinimum, _
SerializationFormatter:=True), _
Assembly: FileIOPermissionAttribute(SecurityAction.RequestOptional, _
Write:="C:\SecureFile.txt"), _
Assembly: PermissionSetAttribute(SecurityAction.RequestMinimum, _
Unrestricted:=True)>
- C. <Assembly: ReflectionPermission(SecurityAction.RequestMinimum, _
ReflectionEmit:=False), _
Assembly: SecurityPermission(SecurityAction.RequestMinimum, _
SerializationFormatter:=False), _
Assembly: FileIOPermissionAttribute(SeurityAction.RequestOptional, _
Write:="C:\SecureFile.txt"), _
Assembly: PermissionSetAttribute(SecurityAction.RequestMinimum, _
Unrestricted:=True)>
- D. <Assembly: ReflectionPermission(SecurityAction.RequestMinimum, _

```
ReflectionEmit:=True), _  
Assembly: SecurityPermission(SecurityAction.RequestMinimum, _  
SerializationFormatter:=True), _  
Assembly: FileIOPermissionAttribute(SecurityAction.RequestMinimum, _
```

Answer: D

Explanation

The SDK provides a tool called PERMVIEW that is useful for verifying that your permission requests are correct. Running PERMVIEW on a compiled assembly will read the permission requests out of the assembly's manifest and display them as shown below.

```
C:\>permview filemover.exe
```

```
Microsoft (R) .NET Framework Permission Request Viewer. Version 1.0.XXXX.0
```

```
Copyright (C) Microsoft Corp. 1998-2001
```

```
minimal permission set:
```

```
<PermissionSet class="System.Security.PermissionSet"  
version="1">
```

```
<IPermission class="System.Security.Permissions.FileIOPermission"  
version="1"
```

```
Unrestricted="true"/>
```

```
</PermissionSet>
```

```
optional permission set:
```

```
<PermissionSet class="System.Security.PermissionSet"  
version="1"
```

```
Unrestricted="true"/>
```

```
refused permission set:
```

```
Not specified
```

The Permissions View tool is used to view the minimal, optional, and refused permission sets requested by an assembly. Optionally, you can use Permview.exe to view all declarative security used by an assembly.

permview [/output *filename*] [/decl] *manifestfile*

Argument Description

manifestfile The file that contains the assembly's manifest. The manifest can be either a standalone file or it can be incorporated in a portable executable (PE) file. The extension for this file will usually be .exe or .dll, but it could also be .scr, or .ocx.

Option Description

/decl Displays all declarative security at the assembly, class, and method level for the assembly specified by *manifestfile*. This includes permission requests as well as demands, asserts, and all other security actions that can be applied declaratively. It does not refer to other assemblies linked to the specified assembly.

/h[elp] Displays command syntax and options for the tool.

/output *filename* Writes the output to the specified file. The default is to display the output to the

console.

/? Displays command syntax and options for the tool.

Remarks

Developers can use Permview.exe to verify that they have applied permission requests correctly to their code.

Additionally, users can run Permview.exe to determine the permissions an assembly requires to execute. For

example, if you run a managed executable and get the error,

"System.Security.Policy.PolicyException: Failed to

acquire required permissions," you can use Permview.exe to determine the permissions

the code in your

executable must receive before it will execute.

Reflection emit is a runtime feature that allows code to create dynamic assemblies,

modules, and types. You can

dynamically create instances of these types to use, or you can use reflection emit to

generate an assembly and

persist it to disk as an .exe file or DLL.

ReflectionPermissionAttribute Class

Allows security actions for ReflectionPermission to be applied to code using declarative security. This class

cannot be inherited.

For a list of all members of this type, see ReflectionPermissionAttribute Members.

System.Object

System.Attribute

System.Security.Permissions.SecurityAttribute

System.Security.Permissions.CodeAccessSecurityAttribute

System.Security.Permissions.ReflectionPermissionAttribute

<AttributeUsage(AttributeTargets.Assembly Or AttributeTargets.Class _

Or AttributeTargets.Struct Or AttributeTargets.Constructor Or _

AttributeTargets.Method)>

<Serializable>

NotInheritable Public Class ReflectionPermissionAttribute

Inherits CodeAccessSecurityAttribute

Remarks

The scope of the declaration that is allowed depends on the SecurityAction that is used.

The security information declared by a security attribute is stored in the metadata of the attribute target and is

accessed by the system at run time. Security attributes are used only for declarative

security. For imperative

security, use the corresponding permission class.

Example

The following example of a declarative attribute shows the correct way to request ReflectionPermission for

ReflectionEmit and states that you must have at least this permission to run your code.

<Assembly: ReflectionPermissionAttribute(SecurityAction.RequestMinimum, _

ReflectionEmit := True)>

'In Visual Basic, you must specify that you are using the assembly scope when making a request.

ReflectionPermission at link time. Demands are typically made in managed libraries (DLLs) to help protect methods or classes from potentially harmful code.

```
<ReflectionPermissionAttribute(SecurityAction.Demand, _  
Unrestricted := True)> Public Class SampleClass
```

Since you do not necessarily have control over what permissions are assigned to the code you write, the

common language runtime provides a mechanism for requesting the permissions that you feel your code must

have in order to run properly. If the code is not granted the required permissions, it will not run. And, because

permission requests are stored in an assembly's manifest, the end user can run a tool to determine what

permissions have been requested by the assembly author and then take the appropriate steps to grant those

permissions if they need the code to run on their machine.

Three types of permission requests are supported:

RequestMinimum: The permissions the code must have to run properly. If these permissions cannot be granted, the code will not be executed.

RequestOptional: The permissions that should be granted if allowed by policy. The runtime will attempt to execute code even if permissions it requests as optional have not been granted.

RequestRefuse: The permissions that code should never be granted. Code will not receive these permissions, even if they would normally be granted to it. This is an extra precaution you can take to prevent your code from being misused. Permission requests can only be made in a declarative fashion and must always be at the assembly level (the assembly is the unit to which permissions are granted by the security system). The following code is a request stating that an assembly must have unrestricted access to the file system in order to function.

```
[assembly:FileIOPermission(SecurityAction.RequestMinimum, Unrestricted=true)]  
<Assembly: FileIOPermission(SecurityAction.RequestMinimum, Unrestricted := True)>  
Public Class FileMover  
'something interesting  
End Class
```

Several requests of the same type can be made, in which case the final permission set requested is the aggregate

of all requests of that type. In the example below RequestMinimum is used twice with different permissions to

state that the assembly must have the ability to use Reflection Emit and perform serialization in order for it to function.

```
[assembly:ReflectionPermission(SecurityAction.RequestMinimum,
ReflectionEmit=true)]
[assembly:SecurityPermission(SecurityAction.RequestMinimum,
SerializationFormatter=true)]
<Assembly: ReflectionPermission(SecurityAction.RequestMinimum, ReflectionEmit :=
True)>
<Assembly: SecurityPermission(SecurityAction.RequestMinimum,
SerializationFormatter := True)>
Public Class CodeGenerator
'something interesting
End Class
```

The same permission can also appear in requests of different types. For instance, the example program at the bottom of this page uses an EnvironmentPermission in each of its three requests (Minimum, Optional, and Refuse). This is useful when a permission encompasses a number of operations and you want to ensure that your assembly has the ability to perform some of those operations while being prevented from performing others. It is important to note that any permission you refuse using RequestRefuse will not be granted to your assembly even if you request that same permission using RequestMinimum. In addition to requesting individual permissions, entire sets of permissions can be requested in a compact fashion. The example below shows two requests: one stating that an assembly must have unrestricted access to the file system in order to function and one stating that it will take any and all other permissions that the security system is willing to grant it.

```
[assembly:FileIOPermission(SecurityAction.RequestMinimum, Unrestricted=true)]
[assembly:PermissionSet(SecurityAction.RequestOptional, Name="FullTrust")]
<Assembly: FileIOPermission(SecurityAction.RequestMinimum, Unrestricted := True)>

<Assembly: PermissionSet(SecurityAction.RequestOptional, Name := "FullTrust")>
Public Class FileMover
'something interesting
End Class
```

The previous example shows how to request a permission set by name, but it is also possible to use a custom permission set representing the exact permissions you want.

QUESTION NO: 7

You are an application developer for Testkingprep.in. Users who are temporary employees are members of a group named TemporaryEmployees. You develop a serviced component named TestkingprepComponent. TestkingprepComponent is part of a COM+ application named MyApplication. TestkingprepComponent is secured by using the SecurityRole attribute for the Employees role.

You need to ensure that members of the TemporaryEmployees group are assigned to the Employees role.

- A. The code Access Security Policy tool.
- B. The Permission View tool.
- C. The Component Services tool.
- D. The Secutil tool.
- E. The Microsoft .NET Framework Configuration tool.

Answer: C

Explanation

The Component Services tool is an administrative console for managing components registered to take advantage of component services. This tool can be found under Administrative Tools, either from the Programs Menu or the Control Panel. Components are grouped into COM+ Applications. New applications can be created by right-clicking the COM+ Applications folder and selecting New->Application. Components can be added to an Application by either dragging and dropping the dll, or by right-clicking the Components folder under the application and selecting New->Component. This will launch the Component Install Wizard. The following screenshot shows the Component Services management console from a Windows .NET Server machine.



Defining Roles for an Application

You determine a security policy for an application by defining the security privileges that it requires. To do this you declare a symbolic level of privilege as a role—that is, define the role for the application—and then assign the role to specific resources within the application. This design is fulfilled when the application is deployed and system administrators populate the role with actual users and user groups. For greater detail, see [Role-](#)

Based Security.

To add a role to an application

1. In the console tree of the Component Services administrative tool, locate the COM+ application to which you want to add the role. Expand the tree to view the folders for the application.
2. Right-click the **Roles** folder for the application, point to **New**, and then click **Role**.
3. In the **Role** dialog box, type the name of the new role in the box provided.
4. Click **OK**.

Note After adding roles to the application, you must be sure to assign the roles to the appropriate components, interfaces, and methods. Otherwise, if role-based security has been chosen and enabled and if roles have been added but not assigned, all calls to the application will fail.

Assigning Roles to Components, Interfaces, or Methods

You can explicitly assign a role to any item within a COM+ application that is visible through the Component Services administrative tool. Doing so ensures that any users that are members of the role will be permitted access to that item and any other items that it contains. For example, if you assign the role "Readers" to a component, any member of "Readers" is allowed access to that component and any interfaces and methods it exposes. "Readers" will show up as an Inherited role for any of those interfaces and methods.

A method is accessible to callers only if you assign a role to it, either by explicitly assigning the role directly to the method or by assigning a role to the method's interface or the method's component, in which case the role will be inherited by the method. If no role is assigned and if access checks are enabled, all calls to the method will fail.

Before you can assign a role, you must define it for the application. All roles defined for the application will appear in the **Roles explicitly set for selected item(s)** window on the **Security** tab for any components, methods, and interfaces within the application.

To assign roles to a component, method, or interface

1. In the console tree of the Component Services administrative tool, locate the COM+ application for which the role has been defined. Expand the tree to view the application's components, interfaces, or methods, depending on what you are assigning the role to.
2. Right-click the item to which you want to assign the role, and then click **Properties**.
3. In the properties dialog box, click the **Security** tab.
4. In the **Roles explicitly set for selected item(s)** box, select the roles that you want to assign to the item.
5. Click **OK**.

Any roles that you have explicitly set for an item will be inherited by any lower-level items it contains and will show up in the **Roles inherited by selected item(s)** window for those items.

QUESTION NO: 8

You are an application developer for Testkingprep.in. You create a Web Forms application to track employee expense report information. Information is entered by each user and stored in a Microsoft SQL Server database. The application uses Integrated Windows authentication with impersonation enabled to communicate with the database. All users are assigned to the DataReader role and the DataWriter role in SQL Server.

The employee expense report form contains client-side validation scripts and additional server controls.

This form is ViewState enabled. All employee expense reports must be approved by the accounting department by using a separate form in the application before payment is made.

You need to unit test the security of the application.

What should you do?

- A. Copy the ViewState information to a text file and attempt to decrypt it.
- B. Test the application from the hosting computer and from the client computers.
- C. Create your own page that mimics the approved page and submit that page to the server.
- D. Sign on as a user in the accounting department and verify that you can approve expense reports.

Answer: C

Explanation

Since we are performing a security test the goal is to attempt to bypass the security mechanisms in place.

Making a submission from an avenue other than what is expected / abnormal is a starting point.

ViewState Maintains the UI State of a Page. The Web is stateless, and so are ASP.NET Pages. They are instantiated, executed, rendered, and disposed on every round trip to the server. As a Web developer, you can add statefulness using well-known techniques like storing state on the server in Session state or by posting a page back to itself.

The primary goal of unit testing is to **take the smallest piece of testable software in the application**, isolate it

from the remainder of the code, and determine whether it behaves exactly as you expect.

Each unit is tested

separately before integrating them into modules to test the interfaces between modules.

Unit testing has proven

its value in that a large percentage of defects are identified during its use.

The most common approach to unit testing requires drivers and stubs to be written. The driver simulates a

calling unit and the stub simulates a called unit. The investment of developer time in this activity sometimes results in demoting unit testing to a lower level of priority and that is almost always a mistake. Even though the drivers and stubs cost time and money, unit testing provides some undeniable advantages. It allows for automation of the testing process, reduces difficulties of discovering errors contained in more complex pieces of the application, and test coverage is often enhanced because attention is given to each unit.

For example, if you have two units and decide it would be more cost effective to glue them together and initially test them as an integrated unit, an error could occur in a variety of places:

- Is the error due to a defect in unit 1?
- Is the error due to a defect in unit 2?
- Is the error due to defects in both units?
- Is the error due to a defect in the interface between the units?
- Is the error due to a defect in the test?

Finding the error (or errors) in the integrated module is much more complicated than first isolating the units, testing each, then integrating them and testing the whole.

Drivers and stubs can be reused so the constant changes that occur during the development cycle can be retested frequently without writing large amounts of additional test code. In effect, this reduces the cost of writing the drivers and stubs on a per-use basis and the cost of retesting is better controlled.

Resources

- NUnit Web site
- NUnit Addin for Visual Studio.NET
- NUnitASP Web site
- TestDriven.com Web site
- XP Programming Web site
- Test Driven Development Group page on Yahoo
- Test Driven Development: By Example by Kent Beck—Book page on Amazon.com
- Refactoring: Improving the Design of Existing Code by Martin Fowler—Book page on Amazon.com
- .TEST from ParaSoft
- Unit Testing Database Code by Richard Dallaway
- Unit Testing Database Code on the DevDaily Web site

QUESTION NO: 9

You are an application developer for Testkingprep.in. You are developing an application that will be used by members of three domain user groups in your company. The user groups are named TestkingprepSales, TestkingprepMarketing,

and AccountManagement. Each of the three user groups will have different permission within the application.

You log on to your development computer by using a domain user account that is a member of only the Domain Users and the Developers domain user groups. On your development computer, your user account is a member of only the local Users group. When you finish developing the application, you need to ensure that the application runs correctly before you send the application to the company's internal software testing department.

How should you test the application?

- A. Select one user from each of the three user groups that will run the application. Deploy the application to the client computer of each of these three users. Test the application on each of the computers.
- B. Deploy the application to a client computer. Ask a domain administrator to place the computer's domain account into all three of the user groups that will run the application. Test the application on the client computer.
- C. Ask a domain administrator to create a domain user account for testing. Place the account in each of the three user groups that will run the application. Test the application, logging on to your computer by using the test domain user account.
- D. Ask a domain administrator to create three domain user accounts for testing. Place one account in each of the three user groups that will run the application. Test the application three times, logging on to your computer by using a different test domain user account for each test.

Answer: D

Explanation

Developers are not normally part of the user groups that will be using custom application they design. So their accounts would not be part of the domain user group(s) with access. Applications that have not been fully tested and approved for release need to be tested in a non-production environment.

There is no need to test from three different computers, user groups/user accounts have access to the application not computer accounts.

Since the groups are domain group, which have different permissions, you cannot use one account to test all three.

All three groups' permissions must be tested and each group will hold specific users.

The permission rule of AGGDLP (Accounts into Global Groups into Domain Local Groups which get permissions) must be adhered to.

Since we need to test the permission and application functionality for the three groups, we need three accounts, one for each group.

QUESTION NO: 10

You are an application developer for Testkingprep.in. You are developing an application that will be used both by company users and by contractors.

Contractors will install the application on their own portable computers. A written company policy prohibits contractors from easily accessing or reviewing the source code of company applications. The file servers that contain the source code for the application are configured so that only company software developers have access. You need to ensure that the contractors cannot easily access the application source code.

What should you do?

- A. Run Dotfuscator Community Edition on each of the application assemblies.
- B. Apply a strong name to each of the application assemblies.
- C. Run the Code Access Security Policy tool for each of the application assemblies before distributing the application.
- D. Use Encrypting File System (EFS) to encrypt the compiled application assemblies.

Answer: A

Explanation

Dotfuscator provides all “normal” obfuscation methodologies in addition to many unique ones. No obfuscation technology is 100 percent secure. As with other obfuscators, Dotfuscator merely makes life more difficult for decompilers and disassemblers; it does not claim nor deliver 100 percent protection. Dotfuscator will rename all possible methods. Dotfuscator can rename all public, private, etc. methods that do not override a method from a non-included class. Renaming is done in such a way as to minimize the string heap, which helps to reduce executable size. PreEmptive's Dotfuscator™ for the Microsoft .NET platform helps protect your program against reverse engineering while making it smaller and more efficient. Dotfuscator has a GUI and command line interface. Dotfuscator Community Edition is accessible directly from the tools menu of Visual Studio.NET 2003.

QUESTION NO: 11

You are an application developer for Testkingprep.in. Each client computer in Testkingprep runs either Microsoft Windows XP Professional or Windows 98. You are developing an application that will be used by all users in Testkingprep. Users log on to their client computers by using a domain user account that is a member of the local Power Users group and the user's computer. You log on to your Windows XP Professional computer by using a domain user account that is a member of the local Administrators group and Power Users group on your computer.

When testing your application, you need to ensure that your tests accurately reflect the production environment in which the application will run.

How should you test the application?

- A. Ask a domain administrator to temporarily remove your domain user account from the local Administrators group on your computer while you are testing the application.
- B. Test the application on your computer.
When testing, log on to the computer by using a domain user account that is a member of only the local Power Users group on your computer.
- C. Deploy the application to a Windows XP Professional computer and a Windows 98 computer.
Log on to each computer by using a domain user account that is a member of only the local Power Users group.
- D. Compile the assemblies of the application from the command line by running the **runas** command and specifying a domain user account that is a member of only the local Power Users group on your computer.

Answer: C

Explanation

To appropriately test your application, it needs to be tested in an environment that mimics the production environment.

In this scenario only Windows XP has a local Power Users group, but Windows 98 systems must be tested since they are deployed in the environment.

QUESTION NO: 12

You are an application developer for Testkingprep.in. You develop a Windows Forms application that connects to a local Microsoft SQL Server database by using the Microsoft .NET Framework Data Provider for SQL Server. The application currently connects to the database by using an account that is a member of the System Administrator role in SQL Server.

You need to ensure that the application can connect to the database by using the user account of the interactive user without providing additional permissions.

What should you do?

- A. Modify the application to activate a SQL Server application role.
- B. Modify the application to use SQL Server integrated security.
- C. Modify the application to send a security token that contains the authentication information in a Kerberos ticket.
- D. Modify the application to use a COM+ security roles.

Answer: B

Explanation

SQL Server provides three different security modes for validating logon information.

Security Mode Description

Standard Security The SQL Server user ID and password must be provided by the

application. No attempt is made to use Windows NT client identification. SQL Servers prior to version 4.2 use this option exclusively.

Windows NT Integrated Security The SQL Server user ID is always taken from the Windows NT domain user ID and password.

Mixed Security Unless the client provides a user ID and password, the SQL Server user ID is taken from the Windows NT domain user ID and password.

Standard Security

Standard security is the default installation option for SQL Server. It provides the simplest security model because security exists independently of the Windows NT domain model. You control the level of access a user has to the database and its objects by setting security options within SQL Server itself. Standard security means the SQL Server uses its own validation process for checking all logon accounts. With standard security, each user provides an additional valid SQL Server logon ID and password. Each SQL Server logon specifies the allowed access to each database and its objects (tables, views, stored procedures, and rules).

The logon accounts are considered valid if they appear in the encrypted *syslogins* table.

Authentication consists

of comparing the provided user name and password against similar information maintained in the SQL Server

database. This is the easiest security model to integrate with IIS.

You must use standard security if you are not using multi-protocol or Named Pipes.

Standard security works for

all network configurations. With standard security, SQL Server does not consider the domain the network logon

accounts are using, and also ignores the Windows NT user name and password scheme.

Standard security is the best choice if your application uses Internet Information Server.

An important part of

this configuration is whether or not an authenticated protocol will be used.

Windows NT Integrated Security

With integrated security, SQL Server leverages Windows NT authentication to validate SQL Server logon

accounts. This allows the user to bypass the standard SQL Server logon process. With this approach, a network

user can access a SQL Server database without supplying a separate logon identification or password because

SQL Server obtains the user and password information from the Windows NT network security process.

Integrated security works for all trusted connections and requires either Named Pipes or Multi-Protocol (with

Named Pipes). Trusted connections can be from other Windows NT, Windows 95, or Windows for Workgroups workstations, and also from Microsoft LAN Manager running under either MS-DOS or Microsoft Windows clients.

SQL Server applications with integrated security benefit from all of the Windows NT security features. This includes domain-wide user accounts, encrypted passwords, password aging, logon auditing, and general user administration. Integrated security requires working closely with the network administrator to grant the necessary access permissions according to your application's security model.

Note If your application uses Internet Information Server, you probably will not use SQL Server integrated security.

You can implement integrated security with SQL Server by creating several Windows NT security groups and granting each group the necessary data access rights. For example, you would grant the SystemUsers group permission to perform SELECT, UPDATE, INSERT, and DELETE using normal application processes.

Similarly, you would grant the SystemAdmin group full SQL Server administrator rights and permissions.

Note Regardless of the SQL Server's logon security mode, ODBC and DB-Library client applications can be configured to always request a trusted connection from the server. The benefit is that with a correctly configured Windows NT account the SQL Executive can connect to remote servers.

Mixed Security

The mixed security mode allows validation by using either standard or integrated security modes. With mixed security mode, SQL Server uses integrated security for all trusted connections. For example, for a connection to a trusted ODBC source, authentication occurs via the Windows NT authentication process. If the integrated mode authentication fails, standard security mode is used requiring the entry of valid SQL Server logon information.

Database Security

One of the most common scenarios for a distributed application involves reading and writing data on a remote database. The dilemma that arises is how to do so securely while maintaining application scalability. Where you choose to manage security in your application will greatly impact, either negatively or positively, the scalability of your application.

To achieve scalability using database connection pooling foregoes having the database manage security. This is because database connection pooling requires the connection string be identical to pool connections. Therefore, you must manage security elsewhere. If you must track database operations on per user basis, consider adding a parameter for user identity to each operation and manually log user actions in the database.

Following the advice above, another issue is how to store the database connection string, which typically contains security credentials, so multiple users can access it without compromising security. Most sample applications demonstrate storing the connection string in the Web.config or global.asax files. However, because these files are plain text files that have limited security, it is not the best location for storing this information.

Should an intruder compromise your Web server's security, these files would be easily accessible. Here are just a few alternatives:

- If using the Web.config file, store the connection string encrypted and then decrypt the connection string in your application code when needed.
- Build a COM+ application using the ServicedComponent Class and store the connection string in the construct string for that component.

When storing sensitive information in the constructor string, you should verify the following:

- Only the appropriate users/groups belong to the Reader role of the System Package.

However,

you must carefully manage COM+ to prevent it from being unable to read its own configuration.

- You have controlled and audited access to the %windows%\Registration folder, where the COM+ configuration database (RegDB) stores its files.

For more information, see ServicedComponent Class.

- Use integrated security to make a trusted connection with SQL Server. This makes it possible for you to use a connection string that eliminates the need for storing a password in the connection string, such as:

```
"Data Source=mySqlServer;Integrated Security=SSPI;Initial Catalog=myDB"
```

There are some drawbacks to using integrated security, most of which you can overcome. Because

integrated security requires a Windows account, it defeats connection pooling if you impersonate each

authenticated principal using an individual Windows account. However, if you instead impersonate a limited

number of Windows accounts, with each account representing a particular role, you can overcome this drawback. Each Windows account must be a domain account with IIS and SQL Server in the same or trusted domains. Alternatively, you can create identical (including passwords) Windows accounts on each machine. After a typical installation, the default security authentication mode is Windows Authentication for SQL Server 2000, which is different from SQL Server 7.0. In SQL Server 7.0, the default authentication mode is Mixed (Windows Authentication Mode and SQL Server Authentication). Windows Authentication is a better security method because of the additional security features it provides, such as secure validation and encryption of passwords, password expiration and auditing. For more information, see Authentication Modes. If you configure SQL Server to use Windows Authentication, you could create one Windows account for read-only operations and another Windows account for read/write operations. You then map each Windows account to a SQL Server login and establish the desired permissions. Using application logic, you then determine which Windows account to impersonate when performing database operations. In SQL Server, you can add any Windows user account as a member of a fixed database role. Each member gains the permissions applied to the fixed database role. For more information, see Managing Permissions. For SQL Server 7.0, integrated security does not work with SQL Server's TCP/IP network library, but uses the named pipes network library instead. As an added security measure, the ConnectionString property of the SqlConnection object does not persist or return the full connection string by default. To do so, you must set Persist Security Info to true

QUESTION NO: 13

You are an application developer for Testkingprep.in. To prevent malicious code from running, a written company policy does not permit developers to log on by using accounts that have more permissions than necessary. Your user account is a member of the Users group and the VS Developers group. You attempt to run an application that requires Administrator-level permissions. You receive an error message that states that permission is denied. You need to be able to run the application. What should you do?

- A. Ask the network administrator to add your user account to the domain Administrators group.
- B. Ask the administrator of your client computer to add your user account to the local Administrators group.
- C. Add the administrator of your client computer to add your user account to the Power Users group.
- D. Run the application by using the **runas** command and specify a user account in the local Administrators group.

Answer: D

Explanation

Run Using a Least-Privileged Account

You should develop applications using a non administrator account. Doing so is important primarily to limit the exposure of the logged on user and to help you to design more secure software. For example, if you design, develop, and test an application while you are interactively logged in as an administrator, you are much more likely to end up with software that requires administrative privileges to run. You should not generally log on using the local administrator account. The account that you use on a daily basis should not be a member of the local Administrators group. Sometimes you might still need an account that has administrative privileges—for example, when you install software or edit the registry. Because the default local administrator account is well known, however, and it is the target of many attacks, create a non-standard administrator account and use this only when it is required.

To create accounts for development

Remove your current user account from the Administrators group if it is a member. Create a new custom administration account using a nonstandard name and strong password.

Use your non-administrator account to logon interactively on a daily basis.

When you need to run a command with administrative privileges, use your custom administration account with the Runas.exe command line utility.

Running Privileged Commands

To run a privileged command, you can use one of the following techniques to temporarily change your security context:

Use the Runas.exe utility from a command line. The following command shows you how to use the Runas.exe utility to launch a command console that runs under your custom administration account.
`runas.exe /user:mymachine\mycustomadmin cmd.exe`

By executing Cmd.exe, you start a new command window that runs under the security context of the user you specify with the /user switch. Any program you launch from this command window also runs under this context.

Use Run As from Windows Explorer. You can right-click an executable file in Windows Explorer and click

Run As. To display this item on Windows 2000, hold the shift key down and then right-click an executable file.

When you click Run As, you are prompted for the credentials of the account you want to use to run the executable file.

Use Run As shortcuts. You can create quick launch and desktop shortcuts to easily run applications using a privileged user account. The following example shows a shortcut that you can use to run Windows Explorer

(Explorer.exe) using the administrator account:

```
%windir%\System32\runas.exe /user:administrator explorer
```

Note If using a non-administrator account proves impractical for your environment, still test your application

or component while running as a least privileged user to catch and correct problems before deploying. For

example, your application might incorrectly require administrator privileges without your realizing it, which

would cause the application to fail when it is deployed in a production environment.

QUESTION NO: 14

You are an application developer for Testkingprep.in. You are developing a three-tier application. You enter sample data to test the application. The following exception is caught by the data layer before the application continues to run.

Cannot set column 'Column1' to 'Text too long for maximum length'.

The value violates the MaxLength limit of this column.

You need to improve the security of the application.

Which two actions should you perform? (Each correct answer presents part of the solution. Choose two)

- A. Increase the maximum length of data characters allowed in the column.
- B. Validate all incoming data character lengths at the business layer.
- C. Modify the data layer to process data above the maximum length.
- D. Modify the user interface to prevent users from entering data above the maximum character length.

Answer: B, D

It is best for the approach that you must validate and cleanse data before it is used and/or store. Rule number two is: *data must be validated as it crosses the boundary between*

untrusted and trusted environments. By definition, trusted data is data you or an entity you explicitly trust has complete control over; untrusted data refers to everything else. In short, any data submitted by a user is initially untrusted data.

TextBox.MaxLength Property

Gets or sets the maximum number of characters allowed in the text box.

Public Overridable Property MaxLength As Integer

The maximum number of characters allowed in the text box. The default is 0, which indicates that the property is not set.

"A validator is a control that checks one input control for a specific type of error condition and displays a description of that problem."

This is an important definition, because it means that you frequently need to use more than one validator control for each input control.

For example, if you want to check whether or not user input in a text box is (a) nonblank, (b) a valid date

between a particular range and (c) less than the date in another text input control, you would want to use three

validators. While this might seem cumbersome, remember that to be helpful to the user, you would want to have

three different text descriptions for all these cases.

The different types of validators are listed as follows:

RequiredFieldValidator Checks that the user has entered or selected anything.

RegularExpressionValidator Checks user input against a regular expression. This allows a wide variety

of checks to be made and can be used for things like ZIP codes and phone numbers.

CompareValidator Compares an input control to a fixed value or another input control. It can

be used for password verification fields, for example. It is also possible to do typed date and number comparisons.

RangeValidator Much like CompareValidator, but can check that the input is between two fixed values.

CustomValidator This allows you to write your own code to take part in the validation framework.

The validator controls are the main elements of the solution. A validator is a visual ASP.NET control that

checks a specific validity condition of another control. It generally appears to the user as a piece of text that

displays or hides depending on whether the control it is checking is in error. It can also be an image, or can even

be invisible and still do useful work. There are five types of validator controls that perform different types of

checks.

Another element in our solution is the ValidationSummary control. Large data entry pages generally have an area where all errors are listed. The ValidationSummary automatically generates this content by gathering it up from validator controls on the page.

The final element is the Page object itself. It exposes the all-important "IsValid" property, which you check in server code to determine if all of the user input is OK.

QUESTION NO: 15

You are an application developer for Testkingprep.in. You develop an application that customers will be able to automate by using Microsoft Visual Basic for Applications (VBA) scripts. The application will be accompanied by sample VBA scripts.

Customers must be able to review the sample VBA scripts. You want customers to be able to automate the installed application by using any of the sample VBA scripts or by creating their own automation scripts. You also want to allow customers to choose not to apply any automation scripts.

You need to distribute the sample VBA scripts with your application in a manner that minimizes security risks for the customer.

What should you do?

- A. On installation, place all the sample VBA scripts in a subfolder of the application's installation folder.
- B. On installation, as the user to choose one sample VBA script to install as the application's automation script.
- C. Do not install the same VBA scripts.
Leave the files in a folder on the installation media.
- D. Encrypt same VBA scripts on the installation media and decrypt the files during installation.

Answer: C

Explanation

Security in deployment is one of the tenets of the Microsoft Trustworthy Computing Security Framework. This section contains articles that will help you understand how to deploy your applications in a secure manner and how to use features like code access security to deploy applications in alternative security contexts.

Sample applications, scripts, etc; do not go through much quality or security review at all, if any. It is best

practice not to have them auto installed with deployments as they can be used against the environment/computer.

Users can always copy what they need. VBAs are very powerful and can be used against systems where they

are installed, especially if they are installed without the user's full understanding of the implications of

installing them. Since they are really templates, they are not required for the installation or for normal application operation in the case. Prime example is MSDE which is part of Microsoft Office 2000 and higher, it is not required and therefore is a separate install.

QUESTION NO: 16

You are an application developer for Testkingprep.in. You create a Web application that is used by all users in the company. The application is hosted on the internet Web server, which is named WebServer.

WebServer has IIS 5.0 installed. The Web application is configured to use Integrated Windows authentication. The Web.Config file specifies that the authentication mode is set to Windows.

The application connects to a Microsoft SQL Server database named DataStore. The database is located on WebServer. The SQL Server computer is configured with SQL Server logs disabled. The database connection code is shown in the following code segment.

```
String myConnStr;  
myConnStr = @"Initial Catalog= ""DataStore"";";  
myConnStr = myConnStr + "Data Source=localhost;Integrated  
Security=SSPI;";  
SqlConnection myConn = new SqlConnection (myConnStr);  
string myInsert;  
myInsert = "INSERT INTO Customer (Customer ID, Name) Values ('123',  
'John Doe')";  
SqlCommand myCmd = new SqlCommand (myInsert);  
myCmd.Connection = myConn;  
myConn.Open ( ) ;  
myCmd.ExecuteNonQuery ( ) ;  
myCmd.Connection.Close ( ) ;
```

When you run the application by using Microsoft Internet Explorer, you receive an error message that reads in part: "Login failed for user WebServer\ASPNET."

You need to ensure that the application can run successfully without prompting the user for a user name and password.

What should you do?

- A. Change the authentication mode in IIS to basic authentication. Update the connection string.
- B. Change the authentication mode in IIS to Anonymous and supply a login ID and Server login account that has access to the database. Update the connection string.
- C. Enable Integrated Windows authentication in Internet Explorer.
- D. Enable impersonation in the Web.Config file.

Answer: D

Explanation

The ASPNet account is local to the webserver, and if your SQL Server is on a remote box then the webserver's

ASPNet account can't be granted permissions to the remote box (because it is a local account).

Thus :

a) change the process model in machine.config (or identity in web.config) to use a domain account

-or-

b) enable impersonation in web.config (do not set a username/password) and set the anonymous internet

user account in IIS MMC Snapin to a domain account. Give the domain account permission to login to the remote SQL Server.

ASP.NET Impersonation

When using impersonation, ASP.NET applications can optionally execute with the identity of the client on

whose behalf they are operating. The usual reason for doing this is to avoid dealing with authentication and

authorization issues in the ASP.NET application code. Instead, you rely on Microsoft Internet Information

Services (IIS) to authenticate the user and either pass an authenticated token to the ASP.NET application or, if

unable to authenticate the user, pass an unauthenticated token. In either case, the ASP.NET application

impersonates whichever token is received if impersonation is enabled. The ASP.NET application, now

impersonating the client, then relies on the settings in the NTFS directories and files to allow it to gain access,

or not. Be sure to format the server file space as NTFS, so that access permissions can be set.

Impersonation is disabled by default. For ASP compatibility, the user must explicitly enable impersonation. If

impersonation is enabled for a given application, ASP.NET always impersonates the access token that IIS

provides to ISAPI extensions. That token can be either an authenticated user token, or the token for the

anonymous user (such as IUSR_MACHINENAME). The impersonation occurs regardless of the type of

authentication being used in the application.

Only application code is impersonated; compilation and configuration are read as the process token. The result

of the compilation is put in the "Temporary ASP.NET files" directory. The account that is being impersonated

needs to have read/write access to this directory. If an application is on a universal naming convention (UNC)

share, ASP.NET will always impersonate the token provided to IIS to access that share unless a configured account is used. If an explicit configured account is provided, ASP.NET will use that account in preference to the IIS UNC token. Applications that do want per-request impersonation can simply be configured to impersonate the user making the request. Impersonation is disabled at the computer level by default and, unless overridden, all the application domains inherit this setting. You can enable impersonation by putting a configuration file in the application root directory. For more information about the ASP.NET configuration system, see ASP.NET Configuration.

As is the case with other configuration directives, this directive applies hierarchically. It is respected by nested applications in the hierarchy, unless explicitly overridden. The default value for this setting is as follows.

```
<impersonation enable="false"/>
```

A minimal configuration file to enable impersonation for an application might look similar to the following example.

```
<!-- Web.config file. -->
<identity impersonate="true"/>
```

There is also name support for running an application as a configurable identity. For example:

```
<identity impersonate="true" userName="contoso\Jane" password="pass"/>
```

This enables the entire application to run as contoso\Jane, regardless of the identity of the request, as long

as the password is correct. This type of impersonation can be delegated to another computer.

You can programmatically read the identity of the impersonated user, as shown in the following example.

```
Dim username As String =
System.Security.Principal.WindowsIdentity.GetCurrent().Name
```

In the preceding example, **userName** and **password** are stored in clear text in the configuration file. Although

IIS will not transmit .config files in response to a user agent request, configuration files can be read by other

means, for instance by an authenticated user with proper credentials on the domain that contains the server. To

help increase security, the identity section supports storage of encrypted **userName** and **password** attributes in

the registry as shown in the following example.

```
userName="registry:HKLM\Software\AspNetIdentity,Name"
password="registry:HKLM\Software\AspNetIdentity>Password"
```

The portion of the string after the keyword **registry** and before the comma indicates the name of the registry

key that ASP.NET opens. The portion after the comma contains a single string value name from which

ASP.NET will read the credentials. The comma is required, and the credentials must be stored in the HKLM

hive. If the configuration format is incorrect, ASP.NET will not launch the worker process and the current

account creation failure code path will be followed.

The credentials must be in REG_BINARY format, containing the output of a call to the Windows API function

CryptProtectData. You can create the encrypted credentials and store them in the registry with the ASP.NET

Set Registry console application(Aspnet_setreg.exe), which uses **CryptProtectData** to accomplish the

encryption. To download Aspnet_setreg.exe, along with the Visual C++ source code and documentation, visit

the Web site www.asp.net and search for "aspnet_setreg".

You should configure access to the key storing the encrypted credentials so that access is provided only to

Administrators and SYSTEM. Because the key will be read by the ASP.NET process running as SYSTEM, you

should set the following permissions:

Administrators:F

SYSTEM:F

CREATOR OWNER:F

ProcessAccount:R

This provides two lines of defense to protect the data:

- The ACL permissions require the identity accessing the data to be an Administrator.

An attacker must run code on the server (CryptUnprotectData) to recover the credentials for the account.

Windows Authentication and SQL Server

Duamish7.0 uses SQL Server authentication to access SQL Server. While this is simple and provides an easy

to understand sample, it is not the most secure option. Duamish 7.0 stores its connection strings in the

Web.config file. This location is readable by any user by default so it is very easy for the secrets (credentials) to be unintentionally disclosed.

A better solution is to use Windows integrated security. First, a Windows identity is needed that is known to

both the Internet Information Services (IIS) server and the SQL Server database. There are two options:

• Domain account

If your servers are part of a domain and you can create a domain account for your application then this is the

best method.

- **Local accounts with synchronized passwords**

If you cannot use a domain account, you can create the same local account on both machines (with the same password) to make it work.

Choose one of the methods above then create an account named Duwamish7_Application (either in the domain or in both machines) and make it part of the Guests group only. This way the application runs with the minimum privileges required.

Next, change the connection string in the Web.config file to remove the explicit credentials and enable the use

of integrated security. For example, if the connection string is the following:

```
Data Source=MYDBSERVER; User Id=Duwamish7_login; Password=*****;
```

```
Initial Catalog=Duwamish7;
```

Then change it to:

```
Data Source=MYDBSERVER;Integrated Security=SSPI;Initial Catalog=Duwamish7.0 ;
```

This change means that the identity the thread is running under is used to access the SQL Server database. By

default, ASP.NET (the aspnet_wp.exe process) runs under the local ASPNET account, however the application

code should run under the Duwamish7_Application account instead. The advantage of running the application

code under the Duwamish7_Application account is that it is a Windows account and you can give it the

appropriate permission for SQL Server.

Next, the account needs to be associated with the Duwamish 7.0 application.

To associate the account with Duwamish 7.0

1. Configure IIS.

a. From the **Start** menu point to **Programs**, then **Administrative Tools**, and click **Internet Services Manager**.

The **Internet Information Services** window opens.

b. On the Tree tab, navigate the tree in search of the Duwamish7 virtual directory. (This virtual

directory location varies depending on the installation.) Once located, right-click **Duwamish7** and select

Properties.

The **Duwamish7 Properties** dialog box appears.

c In the **Duwamish7 Properties** dialog box, choose the **Directory Security** tab. Under

Anonymous access and authentication control, click the **Edit** button.

The **Authentication Methods** dialog box appears.

d. Under **Anonymous access**, click the **Edit** button.

The **Anonymous User** dialog box appears.

- e. Clear **Allow IIS to control password**. This is required so that IIS stores the credentials and has them available when it authenticates against SQL Server.
- f. Under **Username** and **Password**, replace the anonymous user account with Duwamish7_Application and enter its password.
- g. Finally, click **OK** three times to finish.
2. Enable impersonation in Web.config by adding the following line:
3. <system.web>
4. <identity impersonate="true" />

....

The final step is configuring SQL Server to give the Duwamish7_Application account the right permissions in the Duwamish7 database (if you are running the database on a different machine you need to repeat this process on each machine):

To configure SQL Server permissions in the Duwamish7 database

1. Open SQL Server.
 - From the **Start** menu point to **Programs**, then **Microsoft SQL Server**, and click **Enterprise Manager**.
 - The **SQL Server Enterprise Manager** window opens.
 2. Create a SQL login.
 - a. On the **Tree** tab, navigate the tree in search of the **Security** folder beneath the server's name and expand it.
 - b. Right-click the **Logins** folder and select **New Login**.
 - The **SQL Server Login Properties – New Login** dialog box appears.
 - c. In the **Name** box enter Duwamish7 Application.
 - d. Under **Authentication**, locate the **Domain** box and choose the local machine name (e.g. MYDBSERVER) or your domain, depending on where the Duwamish 7_Application account was created.
 - e. Set **Security Access** to: **Grant access**.
 - f. Under **Defaults**, set **Database** to Duwamish7.
 - g. Select the **Database Access** tab and make sure that Duwamish7 is checked in the respective **Permit** fields.
 - h. Under the Database roles for Duwamish7 select the **db_denydatareader** and **db_denydatawriter** permissions to prevent this account from accessing the tables directly.
 3. Delete the Duwamish7_login since it will no longer be used.
 - . Remaining on the **Tree** tab in the **SQL Server Enterprise Manager** window, double-click **Logins** under the **Security** folder.
- The various logins are displayed in the right pane of the **SQL Server Enterprise Manager** window.

a. Locate the entry for Duwamish7_Login and delete it by right-clicking it and choosing **Delete**.

The **Duwamish_Login** dialog box appears. Choose **Yes** to confirm the deletion.

4. Grant Execute permissions to all the stored procedures.

. Remaining on the **Tree** tab in the **SQL Server Enterprise Manager** window, navigate the tree in search

of the **Duwamish7** database located beneath the server name, **Databases** folder.

a. Locate and open **Stored Procedures** in the **Duwamish** database folder and for each non-system stored procedure do the following:

Double-click the stored procedure (for instance, the Account_Add entry).

The **Stored Procedure Properties** dialog box appears.

Click the Permissions button.

The **Object Properties – Duwamish7** dialog box appears.

In the **Permissions** table, locate the **EXEC** column and give the Duwamish7 Application account **EXEC**

permissions by selecting the box.

To verify that the SQL Server permissions are configured appropriately, run the application and use the SQL

profiler to verify that the account being used is Duwamish 7_Application.

The following now happens for every request:

1. An anonymous request comes in, IIS impersonates the account you specified for the anonymous user

(Duwamish7_Application) and passes the request to aspnet_wp.exe. Note that although Duwamish uses

Forms authentication, as far as IIS is concerned all requests are anonymous.

2. Because you configured the application to use impersonation (through the Web.config change),

ASP.NET impersonates the IIS identity (Duwamish7_Application) in the current thread.

3. When the application opens a SQL Connection, it uses integrated Windows authentication to execute the

stored procedures under the Duwamish7_Application login.

Distributed Scenario

When running the distributed scenario, keep in mind that all the changes mentioned above for the Web server

need to be done in the application tier (where the Business Facade runs) instead.

QUESTION NO: 17

You are an application developer for Testkingprep.in. You are developing a Windows Forms application.

You deploy a supporting assembly named TestkingprepAssembly.dll to the global assembly cache. During testing, you discover that the application is prevented from accessing My Assembly.dll. You need to ensure that the application can access TestkingprepAssembly.dll.

What should you do?

- A. Digitally sign the application by using a digital certificate.
- B. Run the `caspol.exe -s` on command from the command line.
- C. Run the Assembly Linker to link `TestkingprepAssembly.dll` to the application.
- D. Modify the security policy to grant the application the Full Trust permission.

Answer: B

Explanation

By default, assemblies installed in the global assembly cache (GAC) run with Full trust. The global assembly cache is a computer-wide storage area for shared assemblies. You can install assemblies

that need to be shared among multiple applications in the global assembly cache.

Installing your assemblies into

the global assembly cache provides the following advantages:

- The global assembly cache provides a centralized location for managing assemblies that need to be

shared among multiple applications.

- Assemblies installed into the global assembly cache support side-by-side execution.

Side-by-side

execution occurs when you have multiple versions of the same assembly present in the global assembly

cache, and different applications need to use the different assembly versions. Because all assemblies

installed into the global assembly cache must be created with strong names, the CLR can differentiate

between the versions of a shared assembly, thereby allowing the appropriate version to be used by the

referencing application.

- The global assembly cache provides a version-aware and publisher-aware assembly store.

However, installing an assembly into the global assembly cache introduces the following issues that are not

encountered with private assemblies:

- Installing an assembly into the global assembly cache requires administrative privileges by default,

because the physical location of the global assembly cache is a subfolder of the Windows directory.

- The shared assembly must be created with a strong name. You cannot install assemblies without strong

names into the global assembly cache because the CLR performs integrity checks on all files that make

up shared assemblies based on the strong name. The cache performs these integrity checks to ensure that

an assembly has not been tampered with after it has been created (for example, to prevent a file from

being changed without the manifest reflecting that change).

- You cannot simply copy an assembly into the global assembly cache and have your applications use it.

The assembly must be installed in the global assembly cache. The preferred (and most common)

approach for installing an assembly in the global assembly cache is to use Windows Installer technology.

Note In a production environment, you should always install assemblies into the global assembly cache with

some mechanism that can maintain a count of the number of references to that assembly. This prevents

premature removal of the assembly from the global assembly cache by the uninstall routine of another

application, which would break your application that relies on that assembly. Windows Installer has very robust

reference counting features and is the recommended way to install assemblies into the global assembly cache.

You can actually install an assembly into the global assembly cache without using Windows Installer

technology, by using the Gacutil.exe utility or by using a drag-and-drop operation to move the assembly into the

Global Assembly Cache folder in Windows Explorer. However, using drag-and-drop operations to move

assemblies into the global assembly cache does not implement any reference counting; therefore, it should be

avoided. If you use the Gacutil.exe tool, you should use the /ir switch, which installs assemblies into the global

assembly cache with a traced reference. These references can be removed when the assembly is uninstalled by

using the /ur switch.

After an assembly is installed into the global assembly cache, you cannot simply copy a new version and have

your application use that updated assembly. As with all strong-named assemblies, applications contain the

strong name (complete with version number) of the assembly that they reference in their own manifests. Instead

of simply copying a new version of the strong-named assembly, you can either recompile against the newer

version or provide binding redirection for the referencing application as well. For shared assemblies that reside

in the global assembly cache, you can achieve this in a number of different ways:

- Redirect assembly versions using publisher policy. You can state that applications should use a newer

version of an assembly by including a publisher policy file with the upgraded assembly.

The publisher

policy file, which is located in the global assembly cache, contains assembly redirection settings. If a publisher policy file exists, the runtime checks this file after checking the assembly's manifest and application configuration file. You should use publisher policies only when the new assembly is backward compatible with the assembly being redirected. New versions of assemblies that claim to be backward compatible can still break an application. When this happens, you can use the following setting in the application configuration file to make the runtime bypass the publisher policy:

```
<publisherPolicy apply="no">.
```

- Redirect assembly versions using your application configuration file. As with strong-named private assemblies, you can specify that your application use the newer version of a shared assembly by putting assembly binding information in your application's configuration file.
- Redirect assembly versions with the machine configuration file. This should not be considered as a first choice for most redirection scenarios, because the machine configuration file overrides all of the individual application configuration files and publisher policies, and applies to all applications.

However, there might be rare cases when you want all of the applications on a computer to use a specific version of an assembly. For example, you might want every application to use a particular assembly version because it fixes a security hole. If an assembly is redirected in the machine configuration file, all of the applications using the earlier version will use the later version.

As far as deployment is concerned, updating shared assemblies is more complex than updating private assemblies—not only do you need to ensure that the upgraded assembly is installed in the global assembly cache, but you also need to ensure that configuration or publisher policy files are also deployed.

Verifying Permissions Granted to Your Assemblies

After you set security policy, you can verify the permissions that are granted to your assemblies by that security policy. The .NET Framework configuration tool contains a wizard that you can use to view the permissions given to an assembly by current security policy.

Code Security and Signing in Components

As a component author, you have three primary concerns when considering security for your components:

- To ensure that your code will run wherever it is deployed.
- To protect your component against unauthorized or malicious use, and in addition, provide safeguards that prevent the exploitation of any bugs in your code.
- To assure the developers and users of your component that your component is authentic, originated from you, and can be trusted.

The .NET Framework security model provides you with the ability to ensure all of these requirements and to provide trustworthy, secure code to your clients.

The common language runtime allows the administrator of a computer to set default permissions for that computer. Simply, the administrator decides what unsafe operations can and cannot be allowed to proceed. On a high-security computer, this may mean that operations required by your component are not allowed to execute.

In that case, your component will fail to work properly.

You can address this concern by requesting permissions. Permissions represent potentially unsafe actions, such as access to the file system or calling native code. You request permissions by adding security attributes to your component. At compile time, these attributes are emitted into metadata in the assembly manifest. When the assembly is loaded, the runtime examines the permission requests and applies the security policy. If the policy allows the permissions to be granted, they will be granted. If policy does not allow the requested permissions to be granted, the administrator has the option of revising the policy to allow your component to run. Requesting the permissions required for your component to function ensures that every computer hosting your component will be fully informed as to the permissions required by the code. For details on requesting permissions for your assembly, see [Requesting Permissions](#). For details about how to add security attributes to your code, see [Adding Security Attributes to Components](#).

You can protect your component from unauthorized or malicious use by insuring that callers to your component have the appropriate permissions to use your component. This can be done through *imperative security checks*, which cause permissions to be checked at run time. Using imperative security checks, you are able to create permission objects that safeguard potentially misused regions of code. By calling the **Demand** method of a permission object, you are able to verify that all callers to your protected code member have the appropriate

permission to access it. If a caller attempts to access a protected code member without permission, a

SecurityException is thrown. For details about how to add imperative security checks, see Adding Imperative

Security Checks to Components. You can also protect code members using security attributes. For details on

protecting your code members with security attributes, see Adding Security Attributes to Components.

Code Signing

When you distribute your component, you will want to be able to assure users that you are the author, that the

code is safe and can be trusted, and that the identity of the assembly can be verified. You accomplish this by

signing your code. You can sign an assembly in two ways:

- Assign your assembly a *strong name*. A strong name consists of the assembly's identity, plus a public

key and a digital signature. A strong name is guaranteed to be a unique name, and thereby ensures the

identity of your assembly. For details, see Strong-Named Assemblies.

Although providing a strong name for your assembly guarantees its identity, it is by no means a guarantee

that the code can be trusted. Put another way, the strong name uniquely identifies and guarantees the

identity of the assembly, but does not guarantee that you wrote it. Nevertheless, a trust decision can

reasonably be made based on strong-name identity alone, provided that the assembly is a well-known

assembly.

- Use a digital certificate, also called *signcode*. To sign your assembly using signcode, you must go to a

third-party authority and prove your identity, whereupon you will obtain a digital certificate that guarantees

you as the originator of the component. For details about strong names and signcode, see Assembly Security

Considerations.

Special Considerations for Custom Controls

It is important to realize that user code that is executed by the designer at design time will always run fully

trusted, even if the project with the user code is located where it would receive less than full-trust at runtime.

For example, say you create a Custom Control that makes API calls every second, and locate the code on a

network share. Another developer on the network might incorporate your custom control into his project. At

design time, the code in the Custom Control will run as fully trusted, and will require no additional permissions.

There are two important consequences of this: First is the fact that you might expose your local machine to a security risk through the network by importing unsecure code. This would only be a concern in the case of a malicious user creating a damaging custom control, followed by a developer mistakenly adding it to his project. A second concern is that applications created by this process may fail to work in a real world environment as a developer may be unaware of what additional permissions may need to be granted. Code security is a complex and important topic. You are encouraged to make a thorough study of it, so that you might better protect yourself and your clients.

Code Access Security Policy Tool (Caspol.exe)

The Code Access Security Policy tool enables users and administrators to modify security policy for the machine policy level, the user policy level, and the enterprise policy level.

caspol [*options*]

Option Description

-addfulltrust *assembly_file*

or

-af *assembly_file*

Adds an assembly that implements a custom security object (such as a custom permission or a custom membership condition) to the full trust assembly list for a specific policy level. The *assembly_file* argument specifies the assembly to add. This file must be signed with a strong name. You can sign an assembly with a strong name using the Strong Name Tool (Sn.exe).

Whenever a permission set containing a custom permission is added to policy, the assembly implementing the custom permission must be added to the full trust list for that policy level. Assemblies that implement custom security objects (such as custom code groups or membership conditions) used in a security policy (such as the machine policy) should always be added to the full trust assembly list.

Caution If the assembly implementing the custom security object references other assemblies, you must first add the referenced assemblies to the full trust

assembly list. Custom security objects created using Visual Basic .NET, the Managed Extensions for C++, and JScript reference either Microsoft.VisualBasic.dll, Microsoft.VisualBasic.dll, or Microsoft.JScript.dll, respectively. These assemblies are not in the full trust assembly list by default. You must add the appropriate

assembly to the full trust list before you add a custom security object. Failure to do so will break the security system, causing all assemblies to fail to load. In this situation, the Caspol.exe **-all -reset** option will not repair security. To repair security, you must manually edit the security files to remove the custom security object.

-s[ecurity] {on | off} Turns code access security on or off. Specifying the **-s off** option does not disable role-based security.

Caution When code access security is disabled, all code access demands succeed. Disabling code access security makes the system vulnerable to attacks by malicious code such as viruses and worms. Turning off security gains some extra performance but should only be done when other security measures have been taken to help make sure overall system security is not breached. Examples of other security precautions include disconnecting from public networks, physically securing computers, and so on.

Remarks

Security policy is expressed using three policy levels: machine policy, user policy, and enterprise policy. The set of permissions that an assembly receives is determined by the intersection of the permission sets allowed by these three policy levels. Each policy level is represented by a hierarchical structure of code groups. Every code group has a membership condition that determines which code is a member of that group. A named permission set is also associated with each code group. This permission set specifies the permissions the runtime allows code that satisfies the membership condition to have. A code group hierarchy, along with its associated named permission sets, defines and maintains each level of security policy. You can use the **-user**, **-customuser**, **-machine** and **-enterprise** options to set the level of security policy. For more information about security policy and how the runtime determines which permissions to grant to code, see Security Policy Management.

Caspol.exe Behavior

All options except **-s[ecurity] {on | off}** use the version of the .NET Framework that Caspol.exe was installed with. If you run the Caspol.exe that was installed with version *X* of the runtime, the changes apply only to that version. Other side-by-side installations of the runtime, if any, are not affected. If you run Caspol.exe from the

command line without being in a directory for a specific runtime version, the tool is executed from the first runtime version directory in the path (usually the most recent runtime version installed). The **-s[ecurity] {on | off}** option is a computer-wide operation. Turning off code access security terminates security checks for all managed code and for all users on the computer. If side-by-side versions of the .NET Framework are installed, this command turns off security for every version installed on the computer. Although the **-list** option shows that security is turned off, nothing else clearly indicates for other users that security has been turned off.

When a user without administrative rights runs Caspol.exe, all options refer to the user level policy unless the **-machine** option is specified. When an administrator runs Caspol.exe, all options refer to the machine policy unless the **-user** option is specified.

Caspol.exe must be granted the equivalent of the **Everything** permission set to function. The tool has a protective mechanism that prevents policy from being modified in ways that would prevent Caspol.exe from being granted the permissions it needs to run. If you try to make such changes, Caspol.exe notifies you that the requested policy change will break the tool, and the policy change is rejected. You can turn this protective mechanism off for a given command by using the **-force** option.

The Assembly Linker is used to create assembly manifests. Compilers and development environments may already provide this capability, so it is often not necessary to use this tool directly. The Assembly Linker will be most useful to developers needing to create a single assembly from multiple components files, such as might be produced with mixed-language development.

The Assembly Linker is also used when adding resources to satellite assemblies. The following command line:

- creates satellite .dll ("WorldCalc.Resources.Dll).
- with a German culture ("/c:de").
- links in resources (in "MyStrings.de.resources").
- giving them the name "MyStrings.de.resources".
- al /out:WorldCalc.Resources.Dll /v:1.0.0.0 /c:de /embed:MyStrings.de.resources,MyStrings.de.resources,Private.

The final parameter indicates whether the resource should be made visible to other assemblies and will normally be set to **"Private"**.

Security Semantics for Applications Written in Visual J#

In the current release, all applications or components written using Visual J# must be fully trusted in order to be able to run. If such an application or component runs in a code group that has not been granted the FullTrust named permission set by the security policy, a security exception is thrown. This applies to new applications written in Visual J# where only the .NET Framework class libraries are used, as well as to existing Visual J++ applications upgraded to Visual J#.

Therefore, consider the following when writing applications using Visual J#:

- All applications run from the local machine in the MyComputer code group are granted the FullTrust

permission set under the default security policy of the .NET Framework. Thus, applications run from the local machine cause no issues.

Applications run from remote locations (such as a network share) in the Internet or Intranet code groups. As

applications run in these code groups are not granted the FullTrust permission set under .NET Framework

default security policy, they fail with a security exception.

A potential workaround is to make applications or components run from remote locations fully trusted. In

order to be considered fully trusted by the .NET Framework security policy, one of the following must be

true:

- The component or application has been signed with a key pair, and the .NET Framework security policy on the computer has been modified to grant the FullTrust named permission set to all

components or applications signed with this key pair.

- The component or application has been signed with an Authenticode certificate, and the .NET

Framework security policy on the computer has been modified to grant the FullTrust named permission

set to all components or applications signed with this Authenticode certificate.

- The .NET Framework security policy on the computer has been modified to grant the FullTrust

named permission set to controls downloaded from the Web site (URL) where the component or application is hosted.

- All managed controls hosted in Internet Explorer run in the Internet or Intranet code groups, and therefore fail with a security exception. The same is true when a Visual J# application is downloaded and run using the code download feature in Internet Explorer.

- When the security policy of the MyComputer code group is modified from the default level of FullTrust to a more restricted permission set, applications started from the local machine are no longer run.

QUESTION NO: 18

You are an application developer for Testkingprep.in. You are conducting a code review of an application that was developed by another developer. The application stores both public data and confidential data. The application stores the data in a file on the hard disk of a user's client computer.

The following code segment manages the writing of all application data to the file. The array named data1 contains the public data, and the array named data2 contains the confidential data.

```
Public void WriteData (DES des, byte[] data1, byte [] data2,
FileStream fsout) { CryptoStream cs = new CryptoStream
( Fsout, des.CreateEncryptor (), CryptoStreamMode.Write);
cs.Write (data1, 0, data1.Length);
cs.Write (data2, 0, data2.Length);
cs.FlushFinalBlock ( ); }
```

You need to improve the response time of this application, without reducing its security. Any changes you make to the WriteData function will be reflected in the code portion for reading data.

What should you do?

A. Replace the code segment with the following code segment.

```
public void WriteData (DES des, byte [] data1, byte [] data2,
FileStream fsout) { fsout.Write(data1, 0, data1.Length);
CryptoStream cs = new CryptoStream (
Fsout, des.CreateEncryptor (), CryptoStreamMode.Write);
cs.Write (data2, 0, data2.Length);
cs.FlushFinalBlock (); }
```

B. Replace the call to the FlushFinalBlock method with the following code segment.

```
int excess = (data1.Length+data.Length) % des.BlockSize;
if (excess > 0) {
byte[] padding = new byte [des.BlockSize – excess];
cs.Write (padding, 0, padding.Length); }
```

C. Modify the application to use asymmetric encryption

D. Call the cs.Write function by using data blocks that have a length equal to the des.BlockSize property.

Repeat the call until all the data is written the file.

Answer: A

Explanation

There is no reason to encrypt public data.

The `CryptoStream.FlushFinalBlock` method already implements padding for you. So, if you think that you need to write any additional code for sending or receiving messages that are shorter than the block length, you probably are not using the `CryptoStream` class correctly.

`CryptoStream.FlushFinalBlock` Method

Updates the underlying data source or repository with the current state of the buffer, then clears the buffer.

```
public void FlushFinalBlock();
```

QUESTION NO: 19

You are an application developer for Testkingprep.in. You are conducting a code review of a Windows Forms application that was developed by another developer.

The application includes a function named

`Logon ()`, which validates a user's credentials, and the function validates those credentials by using a database.

The function returns a value of 0 if the user's password is incorrect, a value of 1 if the user's user ID is incorrect, and a value of 2 if both are correct. Users should receive access to the application only if the function returns a value of 2. A function named `EndApp()` is used to exit the application.

The application must display a message to the user, depending on the result of the `Logon ()` function. The application contains the following code segment.

```
int logonresult = Logon ();
switch (logonresult) {
    case 0:
        MessageBox.Show ("User name is OK, password incorredct.");
        break;
    case 1:
        MessageBox.Show ("User name is incorrect.");
        break;
    default:
        MessageBox.Sho (Welcome!);
        break;
}
if (logonresult != 2) (
    EndApp ();
}
```

You need to improve the security of this code segment while maintaining its functionality. You decide to replace the existing code segment.

Which code segment should you use?

- A.

```
if(Logon() != 2) {
    Console.WriteLine ("Logon error.");
    EndApp ();
}
```
- B.

```
if (Logon() != 2) {
    Console.WriteLine ("Logon error.");
```

```

EndApp ();
}
else {
MessageBox.Show ("Welcome!")
}
C. int logonresult = Logon ();
Switch (logonresult) {
case 0:
MessageBox.Show("User name is OK, Password incorrect.");
EndApp ();
Break;
case 1:
MessageBox.Show("User name is incorrect.");
EndApp ();
break;
default:
Messagebox.Show("Welcome!");
break;
}
D. int logonresult = Logon ();
if(logonresult == 2) {
MessageBox.Show ("Welcome!");
}
else {
MessageBox.Show ("User name or password was incorrect.");
EndApp ();
}
}

```

Answer: B

Explanation

Do not give more information than necessary in error messages. In this case we are specifically telling a potential intruder which part of the attempted login failed, they would then target the portion that is failing. It is best practice to show a single general message for success or failure on logons.

QUESTION NO: 20

You are an application developer for Testkingprep.in. You are responsible for maintaining an application that uses data that is stored in a Microsoft SQL Server database.

The application accepts user input into a variable named userInput and saves the input in a smallint column in a SQL Server table. The application uses a stored procedure to save the data. The application contains the following code segment.

```

//Gets input and ensures input is numeric
userinput = GetUserInput ();
if (userinput < - 32767) {

```

```

MessageBox.Show ("Input is out of range.");
}
else {
//Call stored procedure to save input
SaveToSQL (userinput);
}

```

The SQL Server administrator informs you that several errors have been logged on the SQL Server computer. The errors indicate that the stored procedure attempted to save data that was out of range for the smallint column.

You need to prevent these errors from occurring.

What should you do?

A. Modify the stored procedure so that the input parameter is declared as smallint.

B. Change the data type of the column to int.

C. Replace the code segment with the following code segment.

```

//Gets input and ensures input is numeric
userinput = GetUserInput ();
if (userinput > -32768 || userinput < 32767) {
MessageBox.Show ("Input is out of range.");
}

```

```

else {

```

```

//call stored procedure to save input

```

```

SaveToSQL (userinput);
}

```

D. Replace the code segment with the following code segment.

```

//Gets input and ensures input is numeric
userinput = GetUserInput ();
if (Userinput < 0 || userinput > 32767) {
MessageBox.Show ("Input is out of range.");
}

```

```

else {

```

```

//call stored procedure to save input

```

```

SaveToSQL (Userinput);
}

```

Answer: A

Explanation

int, bigint, smallint, and tinyint

Exact number data types that use integer data.

bigint

Integer (whole number) data from -2^{63} (-9,223,372,036,854,775,808) through $2^{63}-1$ (9,223,372,036,854,775,807). Storage size is 8 bytes.

int

Integer (whole number) data from -2^{31} (-2,147,483,648) through $2^{31} - 1$ (2,147,483,647). Storage size is

4 bytes. The SQL-92 synonym for **int** is **integer**.

smallint

Integer data from -2^{15} (-32,768) through $2^{15} - 1$ (32,767). Storage size is 2 bytes.

tinyint

Integer data from 0 through 255. Storage size is 1 byte.

Remarks

The **bigint** data type is supported where integer values are supported. However, **bigint** is intended for special

cases where the integer values may exceed the range supported by the **int** data type. The **int** data type remains

the primary integer data type in SQL Server.

bigint fits between **smallmoney** and **int** in the data type precedence chart.

Functions will return **bigint** only if the parameter expression is a **bigint** data type. SQL Server will not

automatically promote other integer data types (**tinyint**, **smallint**, and **int**) to **bigint**.

It is best for the approach that you must validate and cleanse data before it is used and/or store.

Rule number two is: *data must be validated as it crosses the boundary between untrusted and trusted*

environments. By definition, trusted data is data you or an entity you explicitly trust has complete control over;

untrusted data refers to everything else. In short, any data submitted by a user is initially untrusted data.

When it comes to SQL statements, all dynamic SQL is bad, and parameterized stored procedures must be used.

Dynamic SQL can be easily compromised and used for SQL injection attacks.

Validate SQL Input and Use Parameter Objects

Validate all the input data that you use in SQL commands. Do not permit the client to retrieve more data than it

should. Also, do not trust user input, and do not permit the client to perform operations that it should not

perform. Doing so helps to lower the risk of SQL injection. By rejecting invalid data early before you issue a

command that has the invalid data, you can improve performance by eliminating unnecessary database requests.

Use **Parameter** objects when you build database commands. When you use **Parameter** objects, each parameter

is automatically type checked. Checking the type is another effective countermeasure you can use to help

prevent SQL injection. Ideally, use **Parameter** objects in conjunction with stored procedures to improve

performance. For more information about using parameters, see "Parameters" later in this chapter.

Using Parameters with Stored Procedures

The following code sample illustrates how to use the **Parameters** collection.

```
SqlDataAdapter myCommand = new SqlDataAdapter("AuthorLogin", conn);
myCommand.SelectCommand.CommandType = CommandType.StoredProcedure;
SqlParameter parm = myCommand.SelectCommand.Parameters.Add(
"@au_id", SqlDbType.VarChar, 11);
parm.Value = Login.Text;
```

In the code sample, the **@au_id** parameter is treated as a literal value and not as code that can be run. Also, **the parameter is checked for type and length. In the code fragment, the input value cannot be longer than 11 characters. If the data does not conform to the type or length that is defined by the parameter, an exception is generated.**

Using stored procedures alone does not necessarily prevent SQL injection. The important thing to do is use parameters with stored procedures. If you do not use parameters, your stored procedures may be susceptible to SQL injection if the stored procedures use unfiltered input. For example, the following code fragment is susceptible to SQL injection.

```
SqlDataAdapter myCommand = new SqlDataAdapter("LoginStoredProcedure '" +
Login.Text + "'", conn);
```

Using Parameters with Dynamic SQL

If you cannot use stored procedures, you can still use parameters with dynamic SQL as shown in the following code fragment.

```
SqlDataAdapter myCommand = new SqlDataAdapter(
"SELECT au_lname, au_fname FROM Authors WHERE au_id = @au_id", conn);
SqlParameter parm = myCommand.SelectCommand.Parameters.Add("@au_id",
SqlDbType.VarChar, 11);
parm.Value = Login.Text;
```

QUESTION NO: 21

You are an application developer for Testkingprep.in. You are developing a Windows-based payroll application that will be used by all payroll administrators in the company. The application has a single executable file that uses a separate assembly to modify payroll data.

You need to design security for your application to ensure that the assembly cannot be called by unauthenticated and unauthorized users.

What should you do?

- A. Run the application by using a user account that has access to the application directory.
- B. Modify the application to validate all user-entered data.
- C. Modify the application to authenticate and authorize user access within each assembly as it is called.

- D. Modify the application to authenticate and authorize user access when each user runs the executable file.
- E. Set the folder-level permissions to the executable file by using directory security.

Answer: C

Explanation:

Code Access Security Using C#

In today's fast growing Information Technology world, security is a major concern.

Security is important not

just to authenticate users but also to authorize their actions. Common business scenarios where we need more

control over security are

1. To restrict the user's access to the application, based on their identity

2. To restrict the user's access to protected resources and certain functions of the application.

3. To verify the identity of the calling program, to check if the caller is trusted or not.

Assembly Design Considerations

One of the most significant issues to consider at design time is the trust level of your assembly's target

environment, which affects the code access security permissions granted to your code and to the code that calls

your code. This is determined by code access security policy defined by the

administrator, and it affects the

types of resources your code is allowed to access and other privileged operations it can perform.

When designing your assembly, you should:

- **Identify privileged code**
- **Identify the trust level of your target environment**
- **Sandbox highly privileged code**
- **Design your public interface**

Identify Privileged Code

Identify code that accesses secured resources or performs security sensitive operations.

This type of code

requires specific code access security permissions to function.

Identify Privileged Resources

Identify the types of resources your assembly needs to access; this allows you to identify any potential problems

that are likely to occur if the environment your assembly ultimately runs in does not grant the relevant code

access security permissions. In this case you are forced either to update code access security policy for your

application if the administrator allows this, or you must sandbox your privileged code.

For more information

about sandboxing, see Chapter 9, "Using Code Access Security with ASP.NET."

Identify Privileged Operations

Also identify any privileged operations that your assembly needs to perform, again so that you know which

code access permissions your code requires at runtime.

Identify the Trust Level of Your Target Environment

The target environment that your assembly is installed in is important because code access security policy may

constrain what your assembly is allowed to do. If, for example, your assembly depends on the use of OLE DB,

it will fail in anything less than a full trust environment.

Full Trust Environments

Full trust means that code has an unrestricted set of code access security permissions, which allows the code to

access all resource types and perform privileged operations, subject to operating system security. A full trust

environment is the default environment for a Web application and supporting assemblies installed on a Web

server, although this can be altered by configuring the `<trust>` element of the application.

Partial Trust Environment

A partial trust environment is anything less than full trust. The .NET Framework has several predefined trust

levels that you can use directly or customize to meet your specific security requirements.

The trust level may

also be diminished by the origin of the code. For example, code on a network share is trusted less than code on

the local computer and as a result is limited in its ability to perform privileged operations.

Supporting Partial Trust Callers

The risk of a security compromise increases significantly if your assembly supports partial trust callers (that is,

code that you do not fully trust.) Code access security has additional safeguards to help mitigate the risk. For

additional guidelines that apply to assemblies that support partial trust callers, see

Chapter 8, "Code Access

Security in Practice." Without additional programming, your code supports partial trust callers in the following

two situations:

- Your assembly does not have a strong name.
- Your assembly has a strong name and includes the

AllowPartiallyTrustedCallersAttribute (APTCA)

assembly level attribute.

Why Worry About the Target Environment?

The trust environment that your assembly runs in is important for the following reasons:

- A partial trust assembly can only gain access to a restricted set of resources and perform a restricted set

of operations, depending upon which code access security permissions it is granted by code access security

policy.

- A partial trust assembly cannot call a strong named assembly unless it includes **AllowPartiallyTrustedCallersAttribute**.
- Other partial trust assemblies may not be able to call your assembly because they do not have the necessary permissions. The permissions that a calling assembly must be able to call your assembly are determined by:

- The types of resources your assembly accesses
- The types of privileged operation your assembly performs

Sandbox Highly Privileged Code

To avoid granting powerful permissions to a whole application just to satisfy the needs of a few methods that

perform privileged operations, sandbox privileged code and put it in a separate assembly. This allows an

administrator to configure code access security policy to grant the extended permissions to the code in the

specific assembly and not to the whole application.

For example, if your application needs to call unmanaged code, sandbox the unmanaged calls in a wrapper

assembly, so that an administrator can grant the **UnmanagedCodePermission** to the wrapper assembly and not the whole application.

Note Sandboxing entails using a separate assembly and asserting security permissions to prevent full stack walks.

For more information about sandboxing unmanaged API calls, see "Unmanaged Code" in Chapter 8, "CodeAccess Security in Practice."

Design Your Public Interface

Think carefully about which types and members form part of your assembly's public interface. Limit the

assembly's attack surface by minimizing the number of entry points and using a well designed, minimal public interface.

Class Design Considerations

In addition to using a well defined and minimal public interface, you can further reduce your assembly's attack

surface by designing secure classes. Secure classes conform to solid object oriented design principles, prevent

inheritance where it is not required, limit who can call them, and which code can call them. The following

recommendations help you design secure classes:

- **Restrict class and member visibility**
- **Seal non base classes**
- **Restrict which users can call your code**
- **Expose fields using properties**

Restrict Class and Member Visibility

Use the **public** access modifier only for types and members that form part of the assembly's public interface.

This immediately reduces the attack surface because only public types are accessible by code outside the

assembly. All other types and members should be as restricted as possible. Use the **private** access modifier

wherever possible. Use **protected** only if the member should be accessible to derived classes and use **internal**

only if the member should be accessible to other classes in the same assembly.

Note C# also allows you to combine **protected** and **internal** to create a **protected internal** member to limit

access to the current assembly or derived types.

Seal Non-Base Classes

If a class is not designed as a base class, prevent inheritance using the **sealed** keyword as shown in the

following code sample.

```
public sealed class NobodyDerivesFromMe
{ }
```

For base classes, you can restrict which other code is allowed to derive from your class by using code access

security inheritance demands. For more information, see "Authorizing Code" in Chapter 8, "Code Access

Security in Practice."

Restrict Which Users Can Call Your Code

Annotate classes and methods with declarative principal permission demands to control which users can call

your classes and class members. In the following example, only members of the specified Windows group can

access the **Orders** class. A class level attribute like this applies to all class members.

Declarative principal

permission demands can also be used on individual methods. Method level attributes override class level

attributes.

```
[PrincipalPermission(SecurityAction.Demand,
Role=@"DomainName\WindowsGroup")]
public sealed class Orders()
{ }
```

QUESTION NO: 22

You are an application developer for Testkingprep.in. You are developing an ASP.NET Web application.

All users in the company use Microsoft Internet Explore 6.0. A group of users is testing the application.

The users report that when an exception occurs, the full exception information is displayed in their Web browsers.

You need to ensure that the full exception information is not displayed when an exception occurs.

What should you do?

- A. Require users to use HTTPS to access the application.
- B. Trap all exceptions and display a generic error message.
- C. Instruct users to enable friendly error messages in Internet Explorer.
- D. Obfuscate the compiled assemblies of the application.
- E. Modify the application's configuration to disable custom errors.

Answer: B

Explanation

Code: Handling Application-Level Errors (Visual Basic)

This example shows how to create an error handler in the Global.asax file that will catch all unhandled

ASP.NET errors in the application — errors not caught with a Try...Catch block or in a page-level error handler.

In the example, the handler transfers control to a generic error page (Errors.aspx), which interprets the error and

displays an appropriate message.

Example

' Handler in Global.asax file

```
Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
```

```
Server.Transfer("Errors.aspx")
```

```
End Sub
```

' Handler in Errors.aspx file

```
Private Sub Page_Load(ByVal sender As System.Object, _
```

```
ByVal e As System.EventArgs) Handles MyBase.Load
```

```
Dim errMessage As String = ""
```

```
Dim appException As System.Exception = Server.GetLastError()
```

```
If (TypeOf (appException) Is HttpException) Then
```

```
Dim checkException As HttpException = _
```

```
CType(appException, HttpException)
```

```
Select Case checkException.GetHttpCode
```

```
Case 403
```

```
errMessage &= "You are not allowed to view that page."
```

```
Case 404
```

```
errMessage &= "The page you have requested can't be found."
```

```
Case 408
```

```
errMessage &= "The request has timed out."
```

```
Case 500
```

```
errMessage &= "The server can't fulfill your request."
```

```
Case 503
```

```
errMessage &= "The server is experiencing a heavy load."
```

```
Case Else
```

```

errMessage &= "The server has experienced an error."
End Select
Else
errMessage &= "The following error occurred<BR>" & _
appException.ToString
End If
Label1.Text = errMessage & "<BR>Please contact the server" & _
"administrator."
Server.ClearError()
End Sub

```

Compiling the Code

This example requires:

- A Web application.
- A Web Forms page called Errors.aspx that has on it a **Label** control called Label1.

Robust Programming

It is preferable to use Try...Catch blocks around any code that is subject to errors rather than rely on a global error handler.

An error handler defined in the Global.asax will only catch errors that apply to ASP.NET. For example, it will

catch the error if a user requests an .aspx file that does not exist in your application.

However, it does not catch

the error if a user requests a nonexistent .htm file. For non-ASP.NET errors, you can create a custom handler in

IIS.

You cannot directly output information from the Global.asax file; you must transfer control to another page,

typically a Web Forms page. When transferring control to another page, use Server.Transfer. This preserves the

current context so that you can get error information from Server.GetLastError.

After handling an error, you must clear it by calling the ClearError method of the Server object

(HttpServerUtility class).

Security

Do not display error information that might help malicious users compromise your application. For details, see

Displaying Safe Error Messages.

Displaying Safe Error Messages

When your application displays error messages, it should not give away information that a malicious user might

find helpful in attacking your system. For example, if your application unsuccessfully tries to log into a

database, it should not display an error message that includes the user name it is using.

There are a number of ways to control error messages:

- Configure the application not to show verbose error messages to remote (application) users. You can

optionally redirect errors to an application page.

- Include error handling whenever practical and construct your own error messages. In your error handler, you can test to see if the user is local and react accordingly.
- Create a global error handler at the page or application level that catches all unhandled exceptions and routes them to a generic error page. That way, even if you did not anticipate a problem, at least users will not see an exception page.

To configure the application to turn off errors for remote users

- In the Web.config file for your application, make the following changes to the customErrors element:

- Set the mode attribute to RemoteOnly (case sensitive). This configures the application to show detailed errors only to local users (you, the developer).
- Optionally include a defaultRedirect attribute that points to an application error page.
- Optionally include <error> elements that redirect specific errors to specific pages. For example, you can redirect standard 404 errors (page not found) to your own application page.

The following example shows a typical customErrors block in the Web.config file.

```
<customErrors mode="RemoteOnly" defaultRedirect="AppErrors.aspx">
<error statusCode="404" redirect="NoSuchPage.aspx"/>
<error statusCode="403" redirect="NoAccessAllowed.aspx"/>
</customErrors>
```

To include error handling

1. Use try-catch-finally blocks around any statements that might generate errors.
2. Optionally, test for a local user with the Context object's UserHostAddress property and modify error handling accordingly. The value 127.0.0.1 is equivalent to "localhost" and indicates that the browser is on the same computer as the Web server.

The following shows an example error-handling block. If an error occurs, a Session state variable is loaded with details about the message and the application then displays a page that can read the Session variable and display the error. (The error is deliberately written to provide no exploitable details to the user.) If the user is local, different error details are provided. In the **finally** block, an open resource is released.

```
try
{
    sqlConnection1.Open();
    sqlDataAdapter1.Fill(dsCustomers1);
}
catch (Exception ex)
{
    if(HttpContext.Current.Request.UserHostAddress == "127.0.0.1")
    { Session["CurrentError"] = ex.Message; }
    else
```

```

{ Session["CurrentError"] = "Error processing page."; }
Server.Transfer("ApplicationError.aspx");
}
finally
{
this.sqlConnection1.Close();
}

```

You can also create an error handler that catches all unhandled exceptions at the page level or for the application as a whole.

To create global error handler

- To create a global handler in a page, create a handler for the Page_Error event. To create an applicationwide error handler, in the Global.asax file, add code to the Application_Error method. These methods are

called if an unhandled exception occurs anywhere in your page or application. You can get information

about the most recent error from the HttpServerUtility.GetLastError method.

Note If you have a global error handler, it takes precedence over error handling specified in the

defaultRedirect attribute of the Web.config customErrors element.

The following shows a sample handler that gets information about the current error, puts it into a Session

variable, and calls a generic error-handling page that can extract and display the error information.

```

protected void Application_Error(Object sender, EventArgs e)
{
Session["CurrentError"] = "Global: " +
Server.GetLastError().Message;
Server.Transfer("lasterr.aspx");
}

```

QUESTION NO: 23

You are an application developer for Testkingprep.in. You are developing a forms-based application that reads files that are named by users of the application. The application contains the following method

```

bool approveFileName (string fileName) {
String docRoot = @"C:\TestkingprepApp\Documents\";
//Your code goes here... Throw an exception if you meet an error.
Return true; }

```

Users of the application must not be allowed to access files that are stored in any location other than the C:\TestkingprepApp\Documents folder.

You need to add code to the method to achieve this goal.

Which code segment or code segments should you use? (Choose all that apply.)

A. fileName=Path.GetFullPath(fileName);

```

B. fileName=fileName.ToUpper ();
C. fileName=fileName.ToLower ();
D. docRoot=docRoot.ToLower ();
E. fileName=docRoot+fileName;
F. if (! FileName.StartsWith (docRoot) )
    throw new ApplicationException (
        "User asked for file in wrong directory");

```

Answer: A, F

Explanation:

Path.GetFullPath Method

Returns the absolute path for the specified path string.

```

public static string GetFullPath(
    string path
);

```

Parameters

path

The file or directory for which to obtain absolute path information

Return Value

A string containing the fully qualified location of *path*, such as "C:\MyFile.txt".

Exceptions

Exception Type Condition

ArgumentException *path* is a zero-length string, contains only white space, or contains one or more invalid characters as defined by InvalidPathChars.

-or-

The system could not retrieve the absolute path.

SecurityException The caller does not have the required permissions.

ArgumentNullException *path* is a null reference (**Nothing** in Visual Basic).

NotSupportedException *path* contains a colon (":").

PathTooLongException The specified path, file name, or both exceed the system-defined maximum length. For example, on Windows-based platforms, paths must be less than 248 characters, and file names must be less than 260 characters.

Remarks

The .NET Framework does not support direct access to physical disks through paths that are device names, such

as "\\.\PHYSICALDRIVE0".

The absolute path includes all information required to locate a file or directory on a system.

The file or directory specified by *path* is not required to exist. For example, if c:\temp\newdir is the current directory, calling **GetFullPath** on a file name such as test.txt returns c:\temp\newdir\test.txt. The file need not

exist. However, if *path* does exist, the caller must have permission to obtain path information for *path*. Note that unlike most members of the Path class, this method accesses the file system.

This method uses current directory and current volume information to fully qualify *path*. If you only specify a file name in *path*, **GetFullPath** returns the fully qualified path of the current directory.

If you pass in a short file name, it is not expanded to a long file name.

For an example of using this method, see the Example section below. The following table lists examples of other typical or related I/O tasks.

To do this... See the example in this topic...

Create a text file. Writing Text to a File

Write to a text file. Writing Text to a File

Read from a text file. Reading Text from a File

Retrieve a file extension. GetExtension

Retrieve only the file name from a path. GetFileNameWithoutExtension

Retrieve only the directory name from a path. GetDirectoryName

Change the extension of a file. ChangeExtension

Sort files in a directory by size. GetFileSystemInfos

Determine if a directory exists. Exists

Determine if a file exists. Exists

Example

The following example demonstrates the **GetFullPath** method on a Windows-based desktop platform.

```
string fileName = "myfile.ext";
string path = @"\"mydir\";
string fullPath;
fullPath = Path.GetFullPath(path);
Console.WriteLine("GetFullPath('{0}') returns '{1}'",
path, fullPath);
fullPath = Path.GetFullPath(fileName);
Console.WriteLine("GetFullPath('{0}') returns '{1}'",
fileName, fullPath);
```

QUESTION NO: 24

You are an application developer for Testkingprep.in. You are developing an application that will be used in the human resources departments of companies that purchase the application. The application includes administrative features that enable user to access and modify confidential information that the application manages. Usability testing indicates that customers deploy the application without properly restricting administrative features.

You need to ensure that the application is more secure when it is deployed.

What should you do?

A. Develop sample template configuration files for the application.

B. Configure the default installation of the application to disable all the administrative

- C. Develop two editions of the application that have the administrative features enabled by default in one edition and disabled by default in the other edition.
- D. Create a Readme.doc file that describes the risks of leaving the administrative features enabled and provides instructions for disabling the features.

Answer: B

Explanation

Security Vision and Framework (SD3+C)

Secure by Design. Implementing threat modeling and other key security considerations in design and

development stages. These considerations include: mandatory training in writing secure code; code reviews and

penetration testing; automated code diagnostic tools; and redesigned architecture to maximize software

resilience.

Secure by Default. Maximizing security in default configurations of shipped software.

To reduce risk of attack,

Microsoft has changed default configurations so that service settings are not enabled at delivery.

Secure in Deployment. Promoting more secure deployment and management of our software. These efforts

include scanning tools, services—including patch management with configuration verification functions, and

localized versions of security bulletins and tools, such as Software Update Services and Baseline Security

Analyzer.

Communications. Keeping customers informed. These efforts include timely communication about software

update releases and our worldwide Security Response Process. In addition, we are working with government, partners, and academia to deliver security education, offer security certification programs for IT professionals,

and conduct consumer protection campaigns worldwide.

[http://www.microsoft.com/downloads/details.aspx?FamilyId=B1418E26-3F3F-464E-8196-](http://www.microsoft.com/downloads/details.aspx?FamilyId=B1418E26-3F3F-464E-8196-DA6954E1E480&displaylang=en)

[DA6954E1E480&displaylang=en](http://www.microsoft.com/downloads/details.aspx?FamilyId=B1418E26-3F3F-464E-8196-DA6954E1E480&displaylang=en)

MSDN TV: Thinking About Security: Secure by Design, Secure by Default, Secure in Deployment and Communications

In this episode, Michael Howard outlines a simple, yet effective way of thinking about how to build secure applications using SD3+C.

<http://msdn.microsoft.com/security/securecode/threatmodeling/default.aspx>

Threat Modeling

To protect your applications from hackers, you have to understand the threats to your applications. Threat

modeling is comprised of three high-level steps: understanding the adversary's view, characterizing the security

of the system, and determining threats. The resources on this page will help you understand the threat modeling process and build threat models that you can use to secure your own applications.

QUESTION NO: 25

You are an application developer for Testkingprep.in. You develop an ASP.NET Web application that writes to an event log named EventLog1. All managers in Testkingprep will run this application. During a test on a manager's client computer, the application fails in the following code segment. (Line numbers are included for reference only.

```
1 Even Log EventLog1 = new EventLog ();
2
3 if ( ! EventLog.SourceExists ("TestkingprepWebApp") ) {
4 EventLog.CreateEventSource ("TestkingprepWebApp", "Application") ;
5 }
6 EventLog1.Source = "TestkingprepWebApp";
7 EventLog1.WriteEntry ("The event occurred.");
```

You need to ensure that event data is written to EventLog1. You want to achieve this goal without granting unnecessary permissions.

What should you do?

A. Insert the following code into the application.

```
string eventLogDir;
eventLogDir = @"C:\%windir%\system32\config\AppEvent.Evt";
FileIOPermission FilePermission =
new FileIOPermission (FileIOPermission Access.AllAccess, ventLogDir);
FilePermission.Assert ();
```

B. Replace line 7 in the code segment with the following line of code.

```
EventLog.WriteEntry ("The event occurred", "MyWebAp",
EventLogEntryType.Warning);
```

C. Grant the managers the Full Control permission for the event log file.

D. Add the aspnet_wp account to the Administrators group.

E. Create the event log source in the installer class of the application.

Answer: E

Explanation

Writing Entries to Event Logs

When you write an entry to an event log, you specify the message you want to write to the log as a string. A message should contain all information needed to interpret what caused the problem and what to do to correct the problem.

There are two ways you can write an entry to the log; both are equally valid. **The most direct way is to**

register an event source with the log to which you want to write, then instantiate a component and set its

Source property to that log, and finally call **WriteEntry**. If you do this, you do not have to set the **Log** property for the component instance; the log is automatically determined when you connect to a source that has already been registered. For more information on registering a source, see Adding Your Application as a Source of Event Log Entries.

The other way to approach this process is to instantiate an **EventLog** component, set its **Source**, **Machine**, and **Log** properties, and call the **WriteEntry** method. In this case, the **WriteEntry** method would determine whether the source already existed and register it on the fly if it did not.

The following conditions must be met in order to successfully write a log entry:

- The source must be registered with the desired log.

Note You must set the **Source** property on your **EventLog** component instance before you can write entries to a log. When your component writes an entry, the system automatically checks to see if the source you specified is registered with the event log to which the component is writing, and calls **CreateEventSource** if needed.

- The message you specify cannot be more than 16K in length.
- Your application must have write access to the log to which it is attempting to write. For more information, see Security Ramifications of Event Logs.

You can specify several parameters when you write an entry, including the type of entry you're making, an ID that identifies the event, a category, and any binary data you want to append to the entry. For more information on the properties associated with an entry, see EventLog Members.

To write an event log entry

1. Instantiate an **EventLog** component. For more information, see Creating EventLog Component Instances.
2. Use the EventLog.CreateEventSource method to register an event source with the log to which you want to write an entry, using a unique string for the source string. Set the **Source** property for the component to the source you registered. For more information, see Configuring EventLog Component Instances. Call the **WriteEntry** method to specify the entry to be written to the log.
3. If Not EventLog.SourceExists("MyApp1") Then
4. EventLog.CreateEventSource("MyApp1", "Application")
5. End If
6. EventLog1.Source = "MyApp1"
7. EventLog1.WriteEntry("This is a simple event log entry")

QUESTION NO: 26

You are an application developer for Testkingprep.in. You create a serviced component named TestkingprepAdmin. TestkingprepAdmin exposes administrative methods for a Testkingprep management application. The declaration for TestkingprepAdmin includes the following code segment.

```
[assembly: ApplicationAccessControl (true)]  
[ComponentAccessControl (true),  
SecurityRole ("Admin")]  
Public class TestkingprepAdmin : ServicedComponent { }
```

You install TestkingprepAdmin on a test computer. You use a test application that runs on the test computer

under a local computer account named Tester. The Tester account is a member of the User group and the

Debugger User group. When the test application class TestkingprepAdmin, you receive the following error

message: "Access is denied."

You want the test application to have access to TestkingprepAdmin. You want to achieve this goal without

granting unnecessary permissions to the Tester account.

What should you do?

- A. Add the Tester account to the local Administrators group.
- B. Add the Tester account to the Admin role of TestkingprepAdmin by using the Component Services tool.
- C. Add the role named TesterRole to TestkingprepAdmin. Add the Tester account to the TesterRole role by using the Component Services tool.
- D. To the beginning of each method exposed by TestkingprepAdmin add the following code segment. If (ContextUtil.IsCallerInRole ("Admin")) {
// Method body here }
- E. To the beginning of each method exposed by TestkingprepAdmin, add the following code segment.
SecurityCallContext context;
Context = SecurityCallContext.CurrentCall;
If (context.IsUserInRole ("Admin", "Tester")) {
// Method body here }

Answer: B

Explanation

By default, a security role in .NET is a Windows® user group. In my example, only if the caller is a member of

the Teller Windows user group will it be allowed to call the OpenAccount method.

The second role-based security model is part of .NET Enterprise Services and is defined in the

System.EnterpriseServices namespace. .NET Enterprise Services are actually the result of integrating COM+

component services with .NET. They offer a wide range of component services essential for any nontrivial application: object pooling; instance management; transactions; concurrency management; security; loosely coupled events; and asynchronous, disconnected calls.

In order to implement the same requirement as in the previous example using .NET Enterprise Services, derive the BankAccount class from ServicedComponent (components that use .NET Enterprise Services are called serviced components), and use the SecurityRole attribute: using System.EnterpriseServices;

```
public class BankAccount : ServicedComponent{
[SecurityRole("Teller")]
public long OpenAccount(){...}
[SecurityRole("Customer")]
[SecurityRole("Teller")]
public long GetBalance(){...}
/* Rest of the implementation */}
```

An Enterprise Services role is defined in the Enterprise Services Catalog in the application that the serviced component is part of. Typically, you would use the Component Services Explorer (also known as the COM+

Explorer) to define these roles, but you can also use the SecurityRole attribute as an assembly attribute to define them:

```
[assembly: SecurityRole("Teller")]
```

```
[assembly: SecurityRole("Customer")]
```

Then use the Component Services Explorer to add users (or user groups) to these roles.

Figure 1 shows the

Bank App application with the Teller and Customer roles.

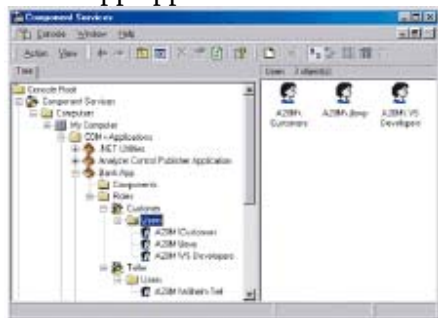


Figure 1 Teller and Customer Roles

QUESTION NO: 27

You are an application developer for Testkingprep.in. You are using the Microsoft .NET Framework to develop an application that uses a Web service. The Web service is provided by a vendor and is accessed over the Internet.

Your application retrieves string data from the Web service and stores it in a variable named webdata.

The application also defines a SqlCommand object named sqlcmd. The application contains the following code segment.

```
String Testkingprepquery;  
Testkingprepquery = "INSERT INTO WebTable (WebData) VALUES (";  
Testkingprepquery += webdata _+ ")" "  
sqlcmd.CommandText = Testkingprepquery;  
sqlcmd.ExecuteNonQuery ();
```

You need to improve the security of this code segment while maintaining its functionality.

What should you do?

- A. Ask the vendor to perform data validation on all data that is provided by the Web service.
- B. Modify the application to use declarative security.
- C. Ask the vendor to provide a Web service that is written by using the .NET Framework.
- D. Format the contents of webdata to be compatible with the SQL Server data type and remove encoded data or SQL statements.

Answer: D

Explanation

It is best for the approach that you must validate and cleanse data before it is used and/or store. Rule number two is: *data must be validated as it crosses the boundary between untrusted and trusted environments*. By definition, trusted data is data you or an entity you explicitly trust has complete control over; untrusted data refers to everything else. In short, any data submitted by a user is initially untrusted data.

When it comes to SQL statements, all dynamic SQL is bad, and parameterized stored procedures must be used.

Dynamic SQL can be easily compromised and used for SQL injection attacks.

All relational databases—including SQL Server, Oracle, IBM DB2, and MySQL—are susceptible to SQL injection

attacks. You can buy products that protect your system from SQL injection, but for most businesses,

the defense against SQL-injection attack must be code-based. The opening for SQL-injection attacks comes

primarily through Web applications that combine user input with dynamic SQL to form SQL commands that the

application sends to the database. Here are four important steps you can take to protect your Web applications

from SQL-injection attacks. In addition to the following tips, the Microsoft Patterns and Practices Library that I

highlighted last month provides advice about securing your data-access applications.

4. Principle of Least Privilege

The account an application uses to connect to the database should have only the privileges that application

requires. The security permissions that an intruder gains from a compromised application define the harm that

the intruder can inflict. Applications shouldn't connect as sa or with the Administrator account. Instead, the account should have permissions to access only the database objects it needs.

3. Validate All Input

If an input field should contain numeric data, then verify that users enter only numbers. If character data is acceptable, check for unexpected characters. Make sure your application looks for characters such as semicolons, equals signs, double dashes, brackets, and SQL keywords. The .NET Framework provides regular expressions that enable complex pattern matching, a good way to test user input. Limiting the length of accepted user input is also a good idea. Validating your input might seem obvious, but many applications are vulnerable to SQL-injection attacks because intruders can use the openings that Web applications offer.

2. Avoid Dynamic SQL

Dynamic SQL is a great tool for performing ad hoc queries, but combining dynamic SQL with user input creates exposure that makes SQL-injection attacks possible. Replacing dynamic SQL with prepared SQL or stored procedures is feasible in most applications. Prepared SQL and stored procedures accept user input as parameter data rather than as SQL commands, thus limiting what an intruder can do. Of course, replacing dynamic SQL with a stored procedure won't help you if you use the user input to build dynamic SQL statements in your stored procedures. In that case, the dynamic SQL that the user input creates will still be corrupted, and your database will still be in danger of SQL-injection attack.

1. Use Double Quotes

Replace all the single quotes that your users' input contains with double quotes. This simple precaution will go a long way toward warding off SQL-injection attacks. Single quotes often terminate SQL expressions and give the input more power than is necessary. Replacing the single quotes with double quotes will cause many SQL injection attacks to fail.

QUESTION NO: 28

You are an application developer for Testkingprep.in. You are developing an application that will include a Windows Service. This Windows Service will monitor a user's computer activities and write records of the monitored activity to data files. The Windows Service will be installed on all computers in Testkingprep. The data files will have a proprietary format and will have a .kbs file name extension. You plan to develop a Windows Forms application that administrators will use to read the contents of the .kbs files. The business rule of your application state the following requirements:

- **Members of the Windows Administrators group must have Full Control permission for .kbs files.**
- **All other users must have no access to .kbs files.**

You need to choose an authorization solution to ensure that only authorized users can access the data files.

What should you do?

- A. Programmatically set the discretionary access control list (DACL) for the files that the application creates.
- B. Create a custom permission for code access by deriving a class from the FileIOPermission class.
- C. Create a custom permission for code access by deriving a class from the CodeAccessPermission class.
- D. Use Microsoft .NET role-based security with WindowsPrincipal objects in your Windows Forms application.

Answer: A

Explanation

Security Descriptors

Windows 2000 implements access control by allowing administrators or owners of objects to assign *security descriptors* to objects stored in Active Directory (or to other types of objects). A security descriptor is a set of information attached to an object (such as a file, printer, or service) that specifies the permissions granted to different groups (or users), as well as the security events to be audited. For example, for the file *temp.dat*, you might grant Read, Write, and Delete permissions to the Administrators group, but assign only Read and Write permissions to the Operators group. For Active Directory objects, in addition to controlling access to a specific object, you can also control access to a specific attribute of that object. For example, you can grant a user access to a subset of information, such as employees' names and phone numbers, but not grant access to the employees' home addresses.

Each security descriptor for an object in Windows 2000 contains four security components:

- **Owner.** By default, the owner is the creator of the object, except for objects created by an administrator, in which case "Administrators" is the owner.
- **Discretionary Access Control List (DACL).** As explained in the Introduction, the DACL (often referred to as ACL) is a list of specific groups, user accounts, and computers that are allowed or denied access to an object. To change a DACL, a permission called WRITE_DAC is required.
- **System Access Control List (SACL).** As explained in the introduction, the SACL specifies which events are

to be audited for which user or group. Examples of events you can audit are file access, logon attempts, and system shutdowns. To read or change the SACL, the **SeSecurityPrivilege** is required.

- **Group (for POSIX).** The Group component is for POSIX compliance and is associated with the "primary group" set in individual user objects in User Manager. (POSIX is based on the UNIX operating system, but it can be implemented by other operating systems.)

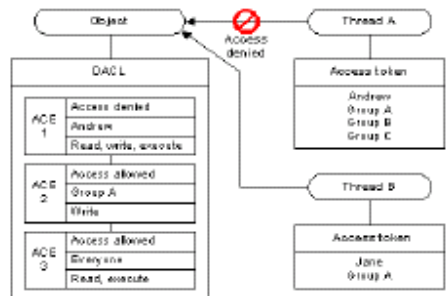
Each assignment of permissions to a group (or user) is known as a permission entry or access control entry (ACE). An ACE is an entry in an access control list (DACL or SACL). The entry contains a SID and a set of access rights. A process (running on behalf of a user) with the user's access token that has a matching security ID is either allowed access rights, denied rights, or allowed rights with auditing. The entire set of permission entries in a security descriptor is known as a permission set.

DACLs and ACEs

If a Windows object does not have a *discretionary access control list (DACL)*, the system allows everyone

full access to it. If an object has a DACL, the system allows only the access that is explicitly allowed by the *access control entries (ACEs)* in the DACL. If there are no ACEs in the DACL, the system does not allow access to anyone. Similarly, if a DACL has ACEs that allow access to a limited set of users or groups, the system implicitly denies access to all trustees not included in the ACEs. In most cases, you can control access to an object by using access-allowed ACEs; you do not need to explicitly deny access to an object. The exception is when an ACE allows access to a group and you want to deny access to a member of the group. To do this, place an access-denied ACE for the user in the DACL ahead of the access-allowed ACE for the group. Note that the order of the ACEs is important because the system reads the ACEs in sequence until access is granted or denied. The user's access-denied ACE must appear first; otherwise, when the system reads the group's access allowed ACE, it will grant access to the restricted user.

The following illustration shows a DACL that denies access to one user and grants access to two groups. The members of Group A get Read, Write, and Execute access rights by accumulating the rights allowed to Group A and rights allowed to Everyone. The exception is Andrew, who is denied access by the access-denied ACE in spite of being a member of the Everyone Group.



QUESTION NO: 29

You are an application developer for Testkingprep.in. You are developing an ASP.NET Web application that will use Forms authentication to authenticate each user who attempts to use the application. The application will store user names and passwords in the <credentials> section of the Web.config file.

You need to configure and implement Forms authentication for the application. Which four actions should you perform? (Each correct answer presents part of the solution. Choose four.)

- A. Add the following element to the Web.config file.
`<authentication mode="Forms">`
`<forms loginUrl="logon.aspx" name="TestkingprepAuthCookie" path="/">`
`</forms>`
`</authentication>`
- B. Add the following element to the Web.config file.
`<authorization>`
`<deny users="?" />`
`<allow users="*" />`
`</authorization>`
- C. Create a Logon.aspx page that includes a Logon button. Add a Click event handler for the Logon button.
- D. Add an Application_OnAuthenticate event handler.
- E. Add code that calls FormsAuthentication.Authenticate to authenticate a user.
- F. Add code that creates an IPrincipal object and associates it with the current HTTP context.

Answer: A, B, C, F

Explanation

Forms Authentication - This is a cookie based authentication system where the username and password is stored in a text file or a database. We will be focusing on this authentication model in this tutorial. However it is the least secure authentication method because your application must handle authentication itself. However, this is the mode you'll be most likely to use on your Internet applications because it requires the least administrative effort to maintain.

Building Secure Microsoft® ASP.NET Applications

Chapter 8: ASP.NET Security

Development Steps for Forms Authentication

The following list highlights the key steps that you must perform to implement Forms authentication:

1. Configure IIS for anonymous access.
2. Configure ASP.NET for Forms authentication.
3. Create a logon Web form and validate the supplied credentials.
4. Retrieve a role list from the custom data store.
5. Create a Forms authentication ticket (store roles in the ticket).
6. Create an **IPrincipal** object.
7. Put the **IPrincipal** object into the current HTTP context.
8. Authorize the user based on user name/role membership.

QUESTION NO: 30

You are an application developer for Testkingprep.in. You are developing a Windows-based application that stores user configuration information for the application. The information is stored in a file named Testkingprep.config. You need to ensure that only users in the Administrators group can make changes to the configuration of the application.

What should you do?

A. Encrypt Testkingprep.config by using the Administrators group's private key. Decrypt the file prior to reading its data from the application.

B. Set a discretionary access control list (DACL) on Testkingprep.config that grants the Administrators group

Read permission and Write permission, but grants other users only Read permission.

C. Add the following code segment to the assembly settings of the application.

```
[assembly: FileIOPermission (SecurityAction.RequestMinimum,  
Read=C:\Program Files\MyApp\Testkingprep.config)]
```

Add the following code segment to the start of the Main() function in the application code.

```
WindowsIdentity wi=WindowsIdentity.GetCurrent ();  
WindowsPrincipal wp=new WindowsPrincipal (wi);  
If (wp.IsInRole (@ "BULITIN\Administrators") ) {  
FileIOPermission fip=new FileIOPermission  
(FileIOPermissionAccess.AllAccess, "Testkingprep.config");  
Fip.Assert (); }
```

D. Add the following code segment at each point that Testkingprep.config is opened.

```
WindowsIdentity wi=WindowsIdentity.GetCurrent ();  
WindowsPrincipal wp = new WindowsPrincipal (wi);  
If (!wp.IsInRole (@ "BUILTIN\Administrators") ) {  
FileIOPermission fip=new  
FileIOPermission(FileIOPermissionAccess.read, "Testkingprep.config");  
Fip.PermitOnly (); }
```

Answer: B

Explanation

Security Descriptors

Windows 2000 implements access control by allowing administrators or owners of objects to assign *security descriptors* to objects stored in Active Directory (or to other types of objects). A security descriptor is a set of information attached to an object (such as a file, printer, or service) that specifies the permissions granted to different groups (or users), as well as the security events to be audited. For example, for the file *temp.dat*, you might grant Read, Write, and Delete permissions to the Administrators group, but assign only Read and Write permissions to the Operators group.

For Active Directory objects, in addition to controlling access to a specific object, you can also control access to a specific attribute of that object. For example, you can grant a user access to a subset of information, such as employees' names and phone numbers, but not grant access to the employees' home addresses.

Each security descriptor for an object in Windows 2000 contains four security components:

- **Owner.** By default, the owner is the creator of the object, except for objects created by an administrator, in which case "Administrators" is the owner.
 - **Discretionary Access Control List (DACL).** As explained in the Introduction, the DACL (often referred to as ACL) is a list of specific groups, user accounts, and computers that are allowed or denied access to an object. To change a DACL, a permission called WRITE_DAC is required.
 - **System Access Control List (SACL).** As explained in the introduction, the SACL specifies which events are to be audited for which user or group. Examples of events you can audit are file access, logon attempts, and system shutdowns. To read or change the SACL, the **SeSecurityPrivilege** is required.
 - **Group (for POSIX).** The Group component is for POSIX compliance and is associated with the "primary group" set in individual user objects in User Manager. (POSIX is based on the UNIX operating system, but it can be implemented by other operating systems.)
- Each assignment of permissions to a group (or user) is known as a permission entry or access control entry

(ACE). An ACE is an entry in an access control list (DACL or SACL). The entry contains a SID and a set of access rights. A process (running on behalf of a user) with the user's access token that has a matching security ID is either allowed access rights, denied rights, or allowed rights with auditing. The entire set of permission entries in a security descriptor is known as a permission set.

QUESTION NO: 31

You are an application developer for Testkingprep.in. You develop an ASP.NET Web application for Testkingprep's intranet. The application accesses data that is stored in a Microsoft SQL Server database.

Access to objects in the database is granted based on the identity of the user of the application. The application uses Windows authentication.

You need to modify the application so that it also uses a new serviced component. The new component requires applications that call it to have membership in the COM+ Authorized Caller role. The developer instruct you to write your application to access the serviced component by using the security context of this user account. You need to modify your code to call the new serviced component by using the security context of the InternalWebAppUser user account.

What should you do?

- A. Disable impersonation in the Web.config file and configure the ASP.NET worker process to run by using the InternalWebAppUser user account.
- B. Set the authentication mode to None in the Web.config file.
- C. Modify the database connection string to connect as the InternalWebAppUser user account.
- D. Write code to impersonate the InternalWebAppUser user account for each call to the serviced component.

Answer: D

Explanation

Securing Your ASP.NET Application and Web Services

Impersonation

By default, ASP.NET applications do not impersonate. The security context of the ASP.NET worker process account (ASPNET by default) is used when your application accesses Windows resources.

A. <identity>

The <identity> element is used to enable impersonation. You can impersonate:

- The original caller (the IIS authenticated identity)
- A fixed identity

Impersonating the Original Caller

To impersonate the original caller, use the following configuration:

<identity impersonate="true" />

The impersonation uses the access token provided by IIS that represents the authenticated caller. This may be

the anonymous Internet user account, for example, if your application uses Forms authentication, or it may be a Windows account that represents the original caller, if your application uses Windows authentication.

If you do enable original caller impersonation, note the following issues:

- Application scalability is reduced because database connections cannot be effectively pooled.
- Administration effort increases as ACLs on back-end resources need to be configured for individual users.
- Delegation requires Kerberos authentication and a suitably configured Windows 2000 environment.

For more information, see "How To: Implement Kerberos Delegation for Windows 2000" in the "How To" section of "Microsoft *patterns & practices* Volume I, *Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication*" at <http://msdn.microsoft.com/library/enus/dnnetsec/html/SecNetHT05.asp>.

Impersonating a Fixed Identity

To impersonate a fixed identity, specify the identity using the **userName** and password attributes on the

<identity> element:

```
<identity impersonate="true" userName="MyServiceAccount"
password="Str0ng!Passw0rd"/>
```

Do not store credentials in plaintext as shown here. Instead, use the `Aspnet_setreg.exe` tool to encrypt the credentials and store them in the registry.

To encrypt credentials for <identity>

1. Run the following command from the command prompt:

2. `aspnet_setreg -k:Software\YourApp\identity -u:CustomAccount :p:StrongPassword`

This stores the encrypted credentials in the specified registry key and secures the registry key with a

restricted ACL that grants Full Control to System, Administrators, and Creator Owner.

3. Reconfigure the **<identity>** element and add the following **userName** and **password** attributes.

4. `<identity impersonate="true"`

5.

`userName="registry:HKLM\SOFTWARE\YourApp\identity\ASPNET_SETREG,userName"`

6.

`password="registry:HKLM\SOFTWARE\YourApp\identity\ASPNET_SETREG,password"/>`

7. Use `Regedt32.exe` to create an ACL on the above registry key that grants read access to the ASP.NET process account.

QUESTION NO: 32

You are an application developer for Testkingprep.in. You develop a library assembly that contains diagnostic utility classes. This library assembly is installed in the global assembly cache on all client computers on the company network.

You develop a Windows Forms application that calls the library assembly. You successfully test the application on your computer, and then you deploy the application to a Web folder on the intranet. Further testing reveals that when you run this application from the intranet, a SecurityException expectation. You need to correct the problem that is causing the SecurityException exception. What should you do?

- A. Add the following the code segment to the library assembly.
[assembly: AllowPartiallyTrustedCallers]
- B. Add the following code segment to the Windows Forms application assembly.
[assembly: allowParticallyTrustedCallers]
- C. Add the following code segment to the library assembly.
[assembly: PermissionSet (SecruityAction.RequestOptional, Name = "LocalIntranet")]
- D. Add the following code segment to the Windows Forms application assembly.
[assembly: PermissionSet (SecurityAction.RequestMinimum, Name="LocalIntranet")]

Answer: D

Explanation

.NET permissions are grouped into NamedPermissionSets. The platform includes the following non-modifiable built-in sets: Nothing, Execution, FullTrust, Internet, LocalIntranet, SkipVerification. The FullTrust set is a special case, as it declares that this code does not have any restrictions and passes any permission check, even for custom permissions. By default, all local code (found in the local computer directories) is granted this privilege.

The above fixed permission sets can be demanded instead of regular permissions:

```
[assembly:PermissionSetAttribute(
SecurityAction.RequestMinimum,
Name="LocalIntranet")]
```

Here is a summary of some facts or rules:

1. If you want to restrict the permissions given to an assembly to only those contained in the associated permission set, you must tick the code group option "The policy level will only have the permissions from the permission set associated with this code group". Otherwise what is granted to the assembly is the permissions of the particular associated permission set plus permissions of the associated permission set of the inherited code group ("All_Code" group).

2. All assemblies must be given “Enable assembly execution” security permission so that it can be run or launched.

3. Permissions included in an assembly’s associated permission set that are above the logged-in user’s privilege will not be granted.

4. **A strongly named assembly can only be called by a fully-trusted caller, unless this assembly states**

AllowPartiallyTrustedCallers. When you use this attribute, it means that you have fully reviewed your

code and there is no security flaw that may be used by luring attackers – such as a improperly used Assert.

Not all system assemblies are marked with this attribute. You can look at the assembly’s manifest to see whether it has that attribute.

5. However, an assembly belonging to the root “All_Code” code group can be called by partially-trusted callers, even if they are strongly named. This is probably because, if you don’t impose a particular security control on an assembly, the runtime security thinks that this assembly is not extremely critical.

6. When you states AllowPartiallyTrustedCallers in an assembly, or let it stay in the “All_Code” code group,

a permission-checking stack walk is still going to be triggered for every attempt to access any controlled resource. The only difference is if you improperly make a Assert you will make luring attacks possible.

QUESTION NO: 33

You are an application developer for Testkingprep.in. You are developing an application that reads the

USERNAME environment variable and executers code in an unmanaged DLL. The design document specifies that the application must display a custom message when the code access security policy restricts access to required resources.

You need to write the code segment that will ascertain whether your application is permitted to access unmanaged code and the USERNAME environment variable.

Your solution must allow the application

to display the custom message when the application is being loaded.

Which code segment should you use?

```
A. try {  
    EnvironmentPermission ep =  
    new EnvironmentPermission (EnvoronmentPermissionAccess.Read,  
    “USERNAME”);  
    Security Permission sp =  
    New SecurityPermission (SecurityPermissionFlag. UnmanagedCode);  
    ep.Demand ();  
    sp.Demant ();  
}
```

```

Catch (SecurityException e) {
// ... }
B. EnvironmentPermission ep =
new EnvironmentPermission (EnvironmentPermissionAccess.Read,
"USERNAME");
SecurityPermission sp =
new SecurityPermission (SecurityPermissionFlag.UnmanagedCode);
if (! (ep.IsUnrestricted () %% sp.IsUnrestricted () ) ) {
// ... }
C. [EnvironmentPermisison (SecurityAction.Demand, Read"USERNAME")]
[SecurityPermission (SecurityAction.Demand, UnmanagedCode=true)]
static void Main () {
// ... }
D. [assembly: EnvironmentPermission (SecirytAction.RequestMinimum, Read=
"USERNAME")]
[assembly: SecurityPermission (SecurityAction.RequestMinimum,
UnmanagedCode=true)]

```

Answer: A

Explanation

Try...Catch...Finally Statements

Provides a way to handle some or all possible errors that may occur in a given block of code, while still running code.

Try

```

[ tryStatements ]
[ Catch [ exception [ As type ] ] [ When expression ]
[ catchStatements ] ]
[ Exit Try ]

```

...

```

[ Finally
[ finallyStatements ] ]

```

End Try

Parts

tryStatements

Optional. Statement(s) where an error can occur. Can be a compound statement.

Catch

Optional. Multiple **Catch** blocks permitted. If an exception occurs while processing the **Try** block, each

Catch statement is examined in textual order to determine if it handles the exception.

Exception

represents the exception that has been thrown.

exception

Optional. Any variable name. The initial value of *exception* is the value of the thrown error. Used with

Catch to specify the error caught.

type

Optional. Specifies the type of class filter. If the value of *exception* is of the type specified by *type* or of a derived type, the identifier becomes bound to the exception object.

When

Optional. A **Catch** statement with a **When** clause will only catch exceptions when *expression* evaluates to **True**. A **When** clause is only applied after checking the type of the exception, and *expression* may refer to the identifier representing the exception.

expression

Optional. Must be implicitly convertible to **Boolean**. Any expression that describes a generic filter.

Typically used to filter by error number. Used with **When** keyword to specify circumstances under which the error is caught.

catchStatements

Optional. Statement(s) to handle errors occurring in the associated **Try** block. Can be a compound statement.

Exit Try

Optional. Keyword that breaks out of the **Try...Catch...Finally** structure. Execution resumes with the

Finally block if present, otherwise with the code immediately following the **End Try** statement. Not

allowed in **Finally** blocks.

Finally

Optional. A **Finally** block is always executed when execution leaves any part of the **Try** statement.

finallyStatements

Optional. Statement(s) that are executed after all other error processing has occurred.

End Try

Terminates the **Try...Catch...Finally** structure.

Remarks

Local variables from a **Try** block are not available in a **Catch** block because they are separate blocks. If you

want to use a variable in more than one block, declare the variable outside the **Try...Catch...Finally** structure.

If errors occur that the programmer has not handled, Visual Studio for Applications simply provides its normal error message to a user, as if there was no error handling.

The **Try** block contains code where an error can occur, while the **Catch** block contains code to handle any error

that does occur. If an error occurs in the **Try** block, program control is passed to the appropriate **Catch**

statement for disposition. The *exception* argument is an instance of the **Exception** class or an instance of a class

that derives from the **Exception** class corresponding to the error that occurred in the **Try** block. The **Exception** class instance contains information about the error including, among other things, its number and message.

In partial trust situations, such as an application hosted on a network share,

Try...Catch...Finally will not catch

security exceptions that occur before the method containing the call is invoked.

EnvironmentPermissionAccess Enumeration

Specifies access to environment variables.

This enumeration has a `FlagsAttribute` attribute that allows a bitwise combination of its member values.

[Visual Basic]

<Flags>

<Serializable>

Public Enum EnvironmentPermissionAccess

[C#]

[Flags]

[Serializable]

public enum EnvironmentPermissionAccess

Remarks

This enumeration is used by `EnvironmentPermission`.

Note Although `NoAccess` and `AllAccess` appear in `EnvironmentPermissionAccess`, they are not valid for use

as the parameter for `GetPathList` because they describe no environment variable access types or all environment

variable access types, respectively, and `GetPathList` expects a single environment variable access type.

Members

Member name Description Value

AllAccess Read and Write access to environment variables. **AllAccess** represents multiple

EnvironmentPermissionAccess values and causes an `ArgumentException` when used as the *flag* parameter for the `GetPathList` method, which expects a single value.

3

NoAccess No access to environment variables.

NoAccess represents no valid

EnvironmentPermissionAccess values and causes an `ArgumentException` when used as the parameter for `GetPathList`, which expects a single value.

0

Read Only read access to environment variables is specified. Changing, deleting and creating environment variables is not included in this

access level.

1

Write Only write access to environment variables is specified. Write access includes creating and deleting environment variables as well as changing existing values. Reading environment variables is not included in this access level. 2

SecurityPermissionFlag.UnmanagedCode Field

UnmanagedCode = 0x2;

Specifies the ability to call unmanaged code.

[Note: Because unmanaged code potentially allows other permissions to be bypassed, this permission should be

used with caution. It is used for applications calling native code using PInvoke.]

Not putting RequestMinimum on the assembly will not provide the necessary security.

All it will ensure is that

the assembly will not load unless it is granted UnmanagedCode permission. It does not put any limitations on

what code can call into this assembly. Now, if the assembly is strong name signed and does not have APTCA

(AllowPartiallyTrustedCallersAttribute), all public methods effectively have an implicit LinkDemand for

FullTrust, so this should be safe. If there is APTCA on the assembly, then the only remedy (apart from completely changing the design) is putting a LinkDemand (or a

Demand) on it and making sure only the right code (for example, signed by the relevant company) can call it

QUESTION NO: 34

You are an application developer for Testkingprep.in. You are developing an ASP.NET Web application that customers will use to order products. Certain customers, named Gold customers, have credit with

Testkingprep. These customers are allowed to place orders without supplying payment information. All other customers must provide credit card information to place an order.

The ASP.NET Web site will use Forms authentication to authenticate all users. The authentication code associated a role named GoldCustomer with the current Web request if the user is a Gold customer.

Your application contains the following method, which returns a value of true if a user enters valid credit card information or a value of false if the user does not enter valid credit card information. (Line numbers are included for reference only.)

```
01 Private bool ValidatePayment Info (string strCCNum, int expMonth, int  
expYear) {
```

```
02 if (strCCNum.Length == 0) {
```

```
03 return false;
```

```
04 }
```

```
05 if (!Regex.IsMatch(strCCNum, m_strCCPattern) ) {
```

```

06 return false;
07 }
08 DateTime dtExpires = new DateTime(expYear, wxpMonth, 1);
09 dtExpires.AddMonths (1);
10 return (DateTime.Now >= dtExpires); }

```

You need to replace the code at line 03 to ensure that the method returns a value of true when the customer is a member of the GoldCustomer role.

Which code segment should you use?

- A. return HttpContext.Current.User.IsInRole ("GoldCustomer");
- B. return HttpContext.Current.User.Identity.Name == "GoldCustomer";
- C. WindowsPrincipal p = new WindowsPrincipal (WindowsIdentity.GetCurrent ()); return p. IsInrole ("GoldCustomer");
- D. Return WindowsIdentity. GetCurrent (). Name == "GoldCustomer";

Answer: A

Explanation

Security Principal Objects

The mechanism that the .NET Framework uses to represent security principals is using the Security Principal

Object provided in the .Net Framework. These are objects which implements a well-known interface named IPrincipal.

A user and his roles are represented in .NET via objects, implementing the IIdentity and IPrincipal interfaces,

attached to the current thread context. IIdentity provides access to name and authentication type information,

and IPrincipal provides access to the contained user identity (one-to-one relationship) and role membership

information. .NET provides two sets of implementations of those interfaces -- WindowsPrincipal with

WindowsIdentity, and GenericPrincipal with GenericIdentity.

The security principal object type depends on the type of authentication used:

WindowsPrincipal for Windows

authentication and GenericPrincipal for everything else

The identity object type also depends on the type of authentication used. Each type of identity object exposes

information specific to an authentication type. WindowsIdentity objects, for example, expose the caller's

Windows access token; PassportIdentity objects expose the caller's PUID; and

FormsIdentity objects expose the caller's forms authentication ticket.

Using IPrincipal and Identity

User data contained in the Security Principal Object can also be accessed through the Page class's User

property. Some sample code is shown below:

```
// Find out whether the caller is authenticated
if (HttpContext.Current.User.Identity.IsAuthenticated) {
// The caller is authenticated
}
// Get an authenticated caller's user name
string name = HttpContext.Current.User.Identity.Name;
// Perform a programmatic role check
if (HttpContext.Current.User.IsInRole ("Managers")) {
// The caller is a manager
}
// Get the caller's access token if using Windows authentication
if (HttpContext.Current.User.Identity is WindowsIdentity) {
IntPtr token =
((WindowsIdentity) HttpContext.Current.User.Identity).Token;
... }
```

QUESTION NO: 35

You are an application developer for Testkingprep.in. You are developing an application that receives signed data. The data is signed by using the RSA encryption algorithm and the SHA1 hash algorithm.

You need to write a function that will verify signatures by using RSA public credentials. Which code segment should you use?

A. public bool VerifySignature (byte [] data, byte[] signature, RSAParameters rsaKey) {
RSACryptoServiceProvider rsa=new RSACryptoServiceProvider ();
Rsa.ImportParameters (rsaKey);
Byte [] mysig=rsa.SignData (data, "SHA1");
For (int I=0; I<mysig.Length; 1++)
If (i>=signature.Length || signature[i] !=mysig [i])
return false;
return true }

B. public bool VerifySignature (byte [] data, byte [] signature, RSAParameters rsaKey) {
RSACryptoServiceProvider rsa=new RSACryptoServiceProvider ();
rsa.importParameters (rsaKey);
return rsa.VerifyData (data, "SHA1", signature); }

C. public bool VerifySignature (byte [] data, byte [] signature, RSAParameters rsaKey) {
RSACryptoServiceProvider rsa=ew RSACryptoServiceProvider ();
rsa.ImportParameters (rsaKey);
byte [] mysig=rsa.Decrypt (data,false);
for (int i=0; i<mysig.Length;i++)
if (i>=signature.Length || signature [i] != mysig [i])
return false;

```

return true; }
D. public bool VerifySignature (byte [] data, byte [] signature,
RSAParameters rsaKey) {

RSACryptoServiceProvider rsa=new RSACryptoServiceProvider ();
Rsa.ImportParameters (rsaKey);
String shaOID=CryptoConfig.MapNameToOID ("SHA1");
Return rsa.VerifyHash (data, shaOID, signature); }

```

Answer: B

Explanation

Verifying Signatures

In order to verify that data was signed by a particular party, you must have the following information:

- **The public key of the party that signed the data.**
- **The digital signature.**
- **The data that was signed.**
- **The hash algorithm used by the signer.**

To verify a signature signed by the RSAPKCS1SignatureFormatter class, use the RSAPKCS1SignatureDeformatter class. The **RSAPKCS1SignatureDeformatter** class must be supplied the

public key of the signer. You will need the values of the modulus and the exponent to specify the public key.

(The party that generated the public/private key pair should provide these values.) First create an

RSACryptoServiceProvider object to hold the public key that will verify the signature, and then initialize an

RSAPParameters structure to the modulus and exponent values that specify the public key.

The following code shows the creation of an **RSAPParameters** structure. The **Modulus** property is set to the

value of a byte array called ModulusData and the **Exponent** property is set to the value of a byte array called

ExponentData.

```
Dim RSAKeyInfo As RSAPParameters
```

```
RSAKeyInfo.Modulus = ModulusData
```

```
RSAKeyInfo.Exponent = ExponentData
```

After you have created the **RSAPParameters** object, you can initialize a new instance of the

RSACryptoServiceProvider class to the values specified in **RSAPParameters**. The

RSACryptoServiceProvider is, in turn, passed to the constructor of an

RSAPKCS1SignatureDeformatter to

transfer the key.

The following example illustrates this process. In this example, HashValue and

SignedHashValue are

arrays of bytes provided by a remote party. **The remote party has signed the HashValue using the SHA1 algorithm, producing the digital signature SignedHashValue.** The **RSAPKCS1SignatureDeformatter.VerifySignature** method verifies that the digital signature is valid and was used to sign the HashValue.

Dim RSA As New RSACryptoServiceProvider()

RSA.ImportParameters(RSAKeyInfo)

Dim RSADeformatter As New RSAPKCS1SignatureDeformatter(RSA)

RSADeformatter.SetHashAlgorithm("SHA1")

If RSADeformatter.VerifySignature(HashValue, SignedHashValue) Then

Console.WriteLine("The signature is valid.")

Else

Console.WriteLine("The signature is not valid.")

End If

The above code fragment will display "The signature is valid" if the signature is valid and "The signature is not valid" if it is not.

RSACryptoServiceProvider.VerifyData Method

Verifies the specified signature data by comparing it to the signature computed for the specified data.

[Visual Basic]

Public Function VerifyData(_

ByVal buffer() As Byte, _

ByVal halg As Object, _

ByVal signature() As Byte _

) As Boolean

Parameters

buffer

The data that was signed.

halg

The name of the hash algorithm used to create the hash value of the data.

signature

The signature data to be verified.

Return Value

true if the signature verifies as valid; otherwise, false.

Remarks

This method verifies the RSA digital signature produced by SignData.

The halg parameter can accept a String, a HashAlgorithm, or a Type.

RSACryptoServiceProvider.VerifyHash Method

Verifies the specified signature data by comparing it to the signature computed for the specified hash value.

[Visual Basic]

Public Function VerifyHash(_

ByVal rgbHash() As Byte, _

ByVal str As String, _

ByVal rgbSignature() As Byte _
) As Boolean

Parameters

buffer

The data that was signed.

halg

The name of the hash algorithm used to create the hash value of the data.

signature

The signature data to be verified.

Return Value

true if the signature verifies as valid; otherwise, false.

Remarks

This method verifies the RSA digital signature produced by SignData.

The halg parameter can accept a String, a HashAlgorithm, or a Type.

QUESTION NO: 36

You are an application developer for Testkingprep.in. You are developing an application that calls a Web service on Testkingprep's intranet. This Web service uses Integrated Windows authentication to authenticate callers. The application must be authenticated by the Web service.

You plan to use a proxy class named ProductInfo to invoke a Web service method named GetProductCount. You write the following code segment. (Line numbers are included for reference only.)

```
1 public int GetInventory (string name) {  
2 ProductInfo productProxy = new Product Info ( ) ;  
3 // ...  
4 return productProxy.GetProductCount (name) ;  
5 }
```

You need to set the credentials that will be used by the call to the Web service method. You need to add code at line 3 of the code segment to achieve this goal. Which code segment should you use?

- A. NetworkCredential cred = new NetworkCredential (_username, _psswd, _domain);
CredentialCache cache = new CredentialCache ();
cache.Add (new Uri(productProxy.Uri), "Basic", cred);
productProxy.Credentials = chache;
- B. NetworkCredential cred = new NetworkCredential (_username, _psswd, _domain);
CredentialCache cache = new CredentialCache ();
cache.add (new Uri (productProxy. Uri), "Windows", cred);
productProxy.Credentials = cache;
- C. productProxy.Credentials = CredentialCache.DefaultCredentials;
- D. GenericIdentity identity = new GenericIdentity (_username, "Windows");
GenericPrincipal principal = new GenericPrincipal (identity, null);

```
Thread.CurrentPrincipal = principal;
```

Answer: B

Explanation

NetworkCredential Class

Provides credentials for password-based authentication schemes such as basic, digest, NTLM, and Kerberos authentication.

For a list of all members of this type, see NetworkCredential Members.

System.Object

System.Net.NetworkCredential

public class NetworkCredential : ICredentials

Any public static (**Shared** in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Remarks

The **NetworkCredential** class is a base class that supplies credentials in password-based authentication

schemes such as basic, digest, NTLM, and Kerberos. Classes that implement the ICredentials interface, such as

the CredentialCache class, return **NetworkCredential** instances.

This class does not support public key-based authentication methods such as SSL client authentication.

Example

The following example associates a **NetworkCredential** instance with a set of Uniform Resource Identifiers

(URIs) in a CredentialCache. It then passes the **CredentialCache** to a WebRequest instance, which uses it to authenticate requests to an Internet server.

```
NetworkCredential myCred = new NetworkCredential(
    SecurelyStoredUserName, SecurelyStoredPassword, SecurelyStoredDomain);
CredentialCache myCache = new CredentialCache();
myCache.Add(new Uri("www.contoso.com"), "Basic", myCred);
myCache.Add(new Uri("app.contoso.com"), "Basic", myCred);
WebRequest wr = WebRequest.Create("www.contoso.com");
wr.Credentials = myCache;
```

QUESTION NO: 37

You are an application developer for Testkingprep.in. Testkingprep runs an e-commerce Web site. Users log on to the Web site by using a password. Passwords are stored in a text file. The following code segment prepares the passwords for storage.

```
Private string HashPassword (string pwd) {
Return FormsAuthentication.HashPasswordForStoringInConfigFile (pwd, "SHA1"); }
```

Users of the Web site are creating passwords that are easily cracked by dictionary attacks.

You need to decrease the likelihood that a dictionary attack will succeed if the password file is stolen, without restricting the passwords that users can create.

What should you do?

A. Create a dictionary file that contains common words. Write additional code to reject passwords that

match the entries the dictionary.

B. Apply a more restrictive discretionary access control list (DACL) to the password storage file.

C. Replace the HashPassword function with the following code segment.

```
Private string HashPassword (string pwd) {  
Return FormsAuthentication.HashPasswordForStoringInConfigFile (pwd,  
"MD5");  
}
```

D. Replace the HashPassword function with the following code segment.

```
Private sting HashPassword (string pwd) {  
RNGCryptoServiceProvider rng=new RNGCryptoServiceProvider ();  
Byte [ ] salt = new byte [16] ;  
rng.GetBytes (salt);  
string saltstr=Convert. ToBase 64String (salt);  
return saltstr +  
FormsAuthentication.HashPasswordForStoringInConfigFile (saltstr +  
pwd, "sha1"); }
```

E. Replace the HashPassword function with the following code segment.

```
Private string HashPassword (string pwe) {  
int hash=0;  
UnicodeEncoding enc=new UnicodeEncoding ();  
byte [] hashdata=enc. GetBytes (pwd);  
  
for (int i=0; i<hashdata.Length; i+=2)  
hash^=hashdata [i] | (hashdata [i+1] <<8);  
return hash.ToString ();  
}
```

Answer: D

Explanation

Hashed passwords are more secure than encrypted passwords because they can't be decrypted. You can harden a hashed password further by adding salt (a cryptographically secure random value) to the hash.

RNGCryptoServiceProvider Class

Implements a cryptographic Random Number Generator (RNG) using the implementation provided by the cryptographic service provider (CSP).

For a list of all members of this type, see RNGCryptoServiceProvider Members.

System.Object

System.Security.Cryptography.RandomNumberGenerator

System.Security.Cryptography.RNGCryptoServiceProvider

public sealed class RNGCryptoServiceProvider :

RandomNumberGenerator

Password Hashing in C#

ASP.NET provides a very convenient way to create a hashed string from a user inputted value. A common issue

with storing passwords in flat file or even the database, is that unwanted eyes can potential see your users

passwords and hack into your web application. In order to hide users passwords in the database, you can create

a hashed value of the password and store it in the database. The benefit of storing a hashed value for your

passwords is that other people will never know the actual password. The drawback of this approach is of course

if you forget the password, its very hard to recover.

How to authenticate using Hashed Passwords?

In normal circumstances when you store passwords in a database in clear text, you simply authenticate a user by

finding the password associated with the username supplied and comparing the inputted password with the

database value. Authentication with hashed passwords is obviously a little different than the traditional

approach. You will create a hash value of the user inputted password and compare it with the already hashed

value in your database. If the two hashed strings are equal, go ahead and authenticate the user.

For example:

```
string strUserInputtedHashedPassword =  
FormsAuthentication.HashPasswordForStoringInConfigFile(  
tbPassword.Text, "sha1");  
if(strUserInputtedHashedPassword ==  
GetUsersHashedPasswordUsingUserName(tbUserName.Text))  
{  
    // sign-in successfull  
}  
else  
{  
    // sign-in failed  
}  
protected TextBox tbPassword;  
protected Literal liHashedPassword;  
private void Page_Load(object sender, System.EventArgs e)  
{
```

```
// Put user code to initialize the page here
}
public void btnHash_Click(object sender, EventArgs e)
{
    if(tbPassword.Text.Length > 0)
    {
        string strHashedPassword =
        FormsAuthentication.HashPasswordForStoringInConfigFile(tbPassword.Text,
        "sha1");
        liHashedPassword.Text = "Hashed Password is: " + strHashedPassword;
    }
}
```

FormsAuthentication.HashPasswordForStoringInConfigFile Method

Given a password and a string identifying the hash type, this routine produces a hash password suitable for storing in a configuration file.

```
public static string HashPasswordForStoringInConfigFile(
    string password,
    string passwordFormat
);
```

Parameters

password

The password to hash.

passwordFormat

The hash algorithm to use. Choices are "sha1" or "md5".

Return Value

Returns a **String** containing a hashed password.

Remarks

Password algorithms supported are SHA1 and MD5.

QUESTION NO: 38

You are an application developer for Testkingprep.in. You are developing an application that will be used by all Testkingprep users. You log on to your development computer by using a user account that has local Administrator permission. However, most company users log on to their client computers by using an account that has only local User permissions. You need to ensure that your testing activities accurately reflect the production environment in which the application will run.

How should you test the application?

- A. Use a test certificate to digitally sign the compiled assemblies of the application.
- B. Remove your user account from the local Administrators group on your development computer.
- C. Add your user account to the local Users group on your development computer.
- D. Run the compiled application from the command line by using the runas command and specifying a user account that has only local User permissions.

Answer: D

Explanation

Least Privilege principle

In every system a number of applications require special privileges in order to perform some system task eg.

system services in *DAC* run with *superuser* privileges. If the sets of privileges associated with such applications

could be made fine-grained enough, as close to minimal needed for the task as possible, unlike in the *DAC*

example above, a damage resulting from a possible vulnerability exploit would be confined to only a portion of

the system accessible by the privileges thus obtained. Therefore, the mandatory security mechanisms of an

operating system should obey the principle of *least privilege* which states that any process in the system should

be allocated only the *absolutely minimal* set of privileges needed to successfully perform the desired task. Any

additional, not needed, privilege possessed by a process increases the damage incurred if a vulnerability in the

program is exploited. Therefore, any mandatory security system should provide scope for implementation of the

least privilege principle.

The principle of least privilege was originally defined by Saltzer (1975):

Every program and every user of the system should operate using the least set of privileges necessary to

complete the job. Primarily, this principle limits the damage that can result from an accident or error. It also

reduces the number of potential interactions among privileged programs to the minimum for correct operation,

so that unintentional, unwanted, or improper uses of privilege are less likely to occur. . .

I sometimes like to think about this principle in reverse. Imagine if you ignore it entirely and run all your code

with full privileges all the time. You've basically turned off a whole raft of security features provided by your

platform. The less privilege you grant to a program, the more walls are erected around that program by the

platform. If a program misbehaves because of a coding error, perhaps one that was tickled by a malicious user

providing malformed input (WhatIsSecureCode), you'll be glad those walls are in place.

Security compromises usually occur in stages: The attacker gains a certain level of privilege via one security

hole and then tries to elevate his privilege level by finding another hole. If you run programs with more

privilege than they really need, the attacker's life is much easier.

This principle can be applied in many different places; it really is a mindset that you should follow as you design and build systems. The following paragraphs describe some examples that are relevant to .NET programmers working on Windows. The examples are by no means exhaustive, but they will give you some concrete ideas that will help you get into the right mindset.

Daemon processes on servers should be designed and configured to run with only the privileges they need to get the job done. This means that you should absolutely avoid the SYSTEM account when configuring the ASP.NET worker process, Windows Services, COM+ servers, and so on ([HowToChooseAnIdentityForADaemon](#)).

Desktop applications should be designed to conform to the Windows Logo guidelines¹ to ensure that they don't attempt to write to protected parts of the file system or registry. When you ship programs that don't follow these guidelines, they break when users attempt to run with least privilege (under normal, nonadministrative user accounts). If you don't want your Mom browsing the Web as an administrator, then start writing programs that she can use as a normal user ([WhatIsANonPrivilegedUser](#))!

When opening files or other secure resources, open them only for the permissions you need for that session. If you plan on reading a file, open it for read-only permissions. Don't open it for read-write permissions thinking, "Someday I may want to write to that file." Open resources for the permission you need at that particular moment.

Use the least privileged form of state management you can for your application. In the .NET Framework, storing application state via Isolated Storage requires less privilege than using a named file, and it has the added benefit of ensuring that your data is written to the user profile ([WhatIsAUserProfile](#)), which is one of the Windows Logo guidelines I alluded to earlier.

Close references to files and other resources as soon as possible. This is especially important if you use impersonation ([WhatIsImpersonation](#)), as these resources can "leak" from one security context ([WhatIsSecurityContext](#)) to the next if you're not careful. Remember that Windows and the .NET Framework tend to check permissions only when resources are first opened. This is a performance enhancement, but it can also lead to security holes if you don't close those resources promptly when you're finished using them.

And, finally, choose to run with least privilege whenever you log in to your computer, whether you're at home or at work. `HowToDevelopCodeAsANonAdmin` provides tips for software developers trying to run with least privilege.

LUA: Least-Privilege User Account

LUA, or *Least-Privilege User Account*, is an acronym I'm sure you'll see repeated over and over in Microsoft presentations from now on. The PDC 2003 presenters often pronounced the acronym as "looah." It's nothing fancy, not even anything really new. LUA refers to the practice of using non-privilege user accounts, both for interactive users and for services.

The Longhorn team wants to simplify security. They believe that there should be two levels of access to the system: least privilege, and administrative. They've even deprecated the Power Users group (referred to in some circles as "admin-lite") in Longhorn.

With the Power Users group gone, application developers really need to make a choice. They need to decide

which of the two levels of privilege (LUA or administrative) their application needs for Longhorn. If the

application doesn't need administrative privileges, then it should be carefully written to work under a leastprivilege

account. That means testing (and, one hopes, even writing the code) using normal user accounts. For

example, a LUA application should write data files in a safe place, such as the user profile, as opposed to the

Program Files directory tree. But what happens to applications that aren't rewritten for Longhorn? What if those

applications don't run well under least-privilege accounts even on Windows XP? A Longhorn feature called

Application Impact Management (AIM) might be able to help those apps run under LUA anyway, as you'll see shortly.

RunAs Class and Control

<http://www.thecodeproject.com/csharp/RunAs.asp>

Developing Software in Visual Studio .NET with Non-Administrative Privileges

[http://msdn.microsoft.com/library/default.asp?url=/library/enus/](http://msdn.microsoft.com/library/default.asp?url=/library/enus/dv_vstechart/html/tchDevelopingSoftwareInVisualStudioNETWithNon-AdministrativePrivileges.asp)

[dv_vstechart/html/tchDevelopingSoftwareInVisualStudioNETWithNon-AdministrativePrivileges.asp](http://msdn.microsoft.com/library/default.asp?url=/library/enus/dv_vstechart/html/tchDevelopingSoftwareInVisualStudioNETWithNon-AdministrativePrivileges.asp)

How can I run Control Panel applets as another user (one with administrative privileges)?

http://www.petri.co.il/run_control_panel_applets_as_another_user.htm

Run Using a Least-Privileged Account

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secmod/html/secmod110.asp>

http://blogs.msdn.com/aaron_margosis/archive/2004/07/07/175488.aspx

QUESTION NO: 39

You are an application developer for Testkingprep.in. You maintain a Windows Forms application. Data entry logic or the application is enforced by the user interface layer. The application contains assemblies that communicate data changes to the database. The application also contains assemblies that implement business logic.

You Create a new assembly named TestkingprepAssembly, which is called from the user interface. Values are passed to TestkingprepAssembly, which performs calculations by using the data. TestkingprepAssembly calls a separate assembly to store the resulting data in a database.

You need to perform unit testing on the application to identify security vulnerabilities caused by unanticipated use of the application.

Which two actions should you perform? (Each correct answer presents part of the solution. Choose two.)

- A. Test the application by calling TestkingprepAssembly directly.
- B. Test the application to verify whether it performs to the original functional specifications.
- C. Test the application by using a domain administrator account.
- D. Test the application by using the account of a user who should not have access to the application.

Answer: A, D

Explanation

Testing for vulnerabilities requires trying to use the application in abnormal ways, passing unacceptable data, etc.

All developers know their code should be tested to improve its quality. However, most developers hate to test.

They especially hate to test their own code, since doing so uses time that instead could be spent writing cool

new features and requires them to face their own potentially imperfect code. One way around these

psychological barriers to testing is to write tests before writing the code, and to make it clear that the code isn't

done until it passes the tests. The tests, in turn, reflect the external requirements of the code's behavior—in essence the tests document the design of the code. Writing tests first as a methodology is known as *test driven development* (or in some circles *test driven design*), or TDD.

Improved Productivity

A key principle of TDD is that developers are most productive when they are in the process of fixing a bug in

an application (note: not looking for the bug). This is one of the few times during development that measurable

progress is being made on the project. While developers spend time on other things, like finding bugs, they aren't able to spend as much time writing code to correct defects (for example, fixing bugs). A key benefit of test driven development (TDD) is to maximize the time developers spend in bug-fix mode, thus maximizing their productivity.

Improved Quality

In TDD, all non-trivial features of an application are tested. These tests are written before the actual code is

written, and they are run all the time. Whenever a test fails, the developer immediately corrects the failure. So, when the time comes to add new functionality, the developer assumes the functionality exists and writes a test.

He then runs the tests (which often won't even compile if they rely on as-yet-undefined classes and methods),

and when they fail, he enters bug-fix mode and begins adding code to the application until he can get all tests to

pass once more. Often, the process of writing the test for the functionality will provide the developer with a

better understanding of how the production code should be designed, resulting in improved final code quality.

TDD also makes code easier to maintain, upgrade, and redesign, and has other benefits which the reader is

encouraged to research.

NUnit: A Test Framework for .NET

NUnit is an open-source testing framework for .NET, modeled after other similar testing frameworks in the

xUnit family, such as JUnit for Java (in fact the initial version was a direct port of JUnit).

You can download

the latest version of NUnit from NUnit.

NUnit provides a simple way for developers to write unit tests of the .NET classes, and comes with a small,

sample application that demonstrates its use. The current version is written in C# and relies on attributes rather

than inheritance and/or naming conventions to define tests and test suites. The main attributes involved are

TestFixture, which applies to a class containing tests, **SetUp** and **TearDown**, which are run before and after each test, and **Test**.

NUnit also provides a simple graphical user interface that lets you select which assembly you want to test and

which set of tests within that assembly you want to run. It then runs all of the tests in the assembly (or

namespace or class) selected, displaying a green bar if everything passes and a red bar if any tests failed. Details

of each failed test are also displayed, making it very easy to locate the cause of the failure.



Figure 1. NUnit's GUI tool provides instant feedback that everything is running as it should be, according to the tests defined so far.

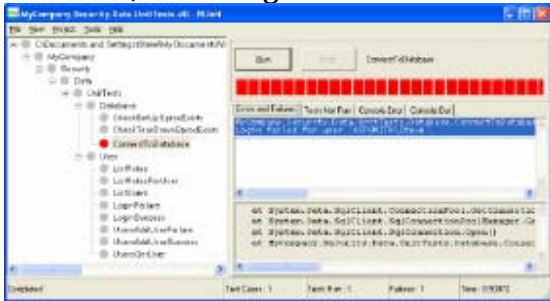


Figure 2. NUnit shows the details of a failure caused by an invalid connection string.

There are other testing tools coming to market for .NET, but today NUnit has the largest support base, and is both free and open source, making it something you may want to try first.

Related Tools

NUnit Visual Studio .NET Addin

This addin, which requires NUnit, allows you to run your tests without leaving the Visual Studio .NET IDE.

Simply right-click on a project or solution in the Solution Explorer to run all tests within that project or solution.

The results are displayed in the Output window. Unfortunately, there are no pretty green or red bars, but if

having one less program open is important to you, this can be a useful tool.

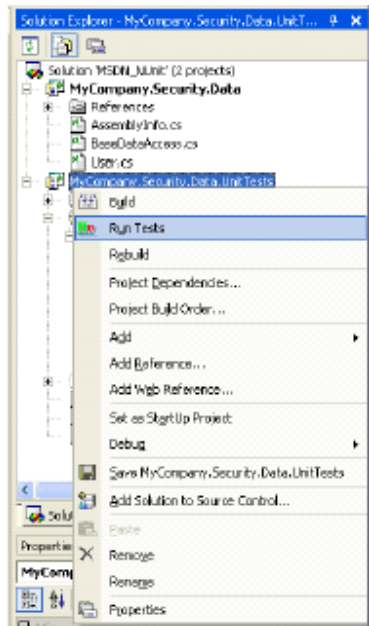


Figure 3. Run tests in Visual Studio .NET with a simple right-click.

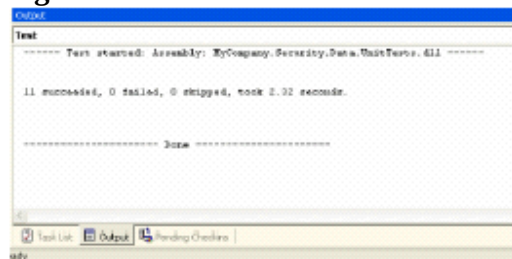


Figure 4. Output from NUnit tests run within Visual Studio .NET

NUnitASP

NUnitASP allows TDD to be extended to the user interface in ASP.NET, by providing a means of testing Web controls. Its usage is beyond the scope of this article, but you may learn more about it from the NUnitASP Web site.

A Simple Application

To demonstrate TDD, I've used it to help me redevelop a multi-tier sample application for ASP.NET. The

application demonstrates how to register a user for an application, how to grant them access to secure web

pages, and how to allow them to sign in and sign out. The data access layer for this application is concerned

with adding users to the database, retrieving user information, and verifying that user logins are correct.

I've used this sample for a couple of years now and built it with TDD several times, learning new things

with each iteration.

Lessons Learned

Use a Test Database

This should really go without saying, but just to be sure, you should always run your tests against a test database that is completely separate from your production database. The schema should match the production database's schema as closely as possible. The database should contain no data, except perhaps static lookup table data like lists of US states or city-ZIP code tables. All of the data you'll be testing will be added during the setup of each test.

Database Setup

Set up the test database using a stored procedure. Originally, I was running the tests using setup routines in the code. At first this resulted in the database changing slightly with every test (most notably, IDENTITY columns were growing ever larger). Eventually, I coded enough into the **SetUp** and **TearDown** routines in my test code to get the database back to a standard repeatable state, but then I found that I was duplicating a lot of this code within each **TestFixture** class. Since it was all SQL script, I moved it into a pair of stored procedures, and I

found that this made for a much cleaner set of test fixtures. The following code sample demonstrates how to set

up simple **test_SetUp** and **test_TearDown** stored procedures, with code for resetting IDENTITY columns. The

DBCC CHECKIDENT function is key to putting IDENTITY columns back to their original states. **Code sample. SetUp and TearDown stored procedures ensure tests run in a consistent, realistic environment.**

```
CREATE PROCEDURE dbo.test_SetUp
```

```
AS
```

```
BEGIN
```

```
exec test_TearDown
```

```
-----  
-- SET UP USERS TABLE  
-----
```

```
exec usp_InsertUser 'John', 'Doe', 'jdoe@test.com', 'password', 0
```

```
exec usp_InsertUser 'Jane', 'Smith', 'jsmith@test.com', 'password', 0
```

```
exec usp_InsertUser 'Admin', 'User', 'admin@test.com', 'password', 0  
-----
```

```
-- SET UP ROLES TABLE  
  
-----
```

```
insert Roles (RoleName) values ('Admins')
```

```
insert Roles (RoleName) values ('Power Users')
```

```
insert Roles (RoleName) values ('Editors')  
-----
```

```
-- SET UP USERROLES TABLE
```

```

-----
-- John Doe
insert UsersInRoles (UserID, RoleID) values (1,1)
insert UsersInRoles (UserID, RoleID) values (1,2)
-- Jane Smith
insert UsersInRoles (UserID, RoleID) values (2,3)
-- Admin User
insert UsersInRoles (UserID, RoleID) values (3,1)
END

```

GO

By using the MS Data Access Application Block, the amount of code required to call these procedures is minimized (one line per call). Figure 5 shows an example **TestFixture** for my **Users** class, which has methods for adding users to and retrieving them from the database. A test user is defined with private variables and used in each of the tests that require an existing user (this user is added by the **test_SetUp** stored procedure).

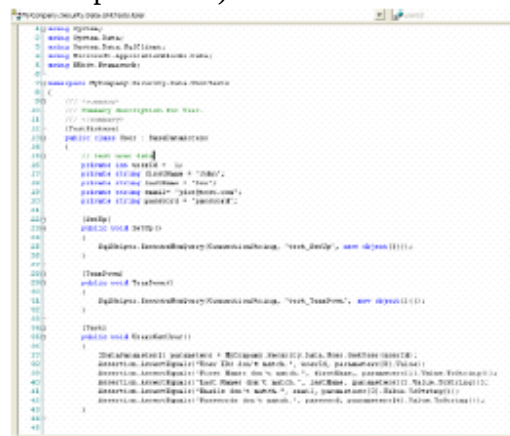


Figure 5. A sample TestFixture

Notice that **SetUp** and **TearDown** are each only one line of code. In fact, since these are the same for all tests against this database (not just the ones in this class), I could probably just move these to my test's base class, which defines its **ConnectionString** property. Also note that I'm calling **test_SetUp** in both **SetUp** and **TearDown**—this is so my front-end application, which uses the same test database, has some data it can use. It would be more efficient to have a totally separate test database, which was completely emptied in each

TearDown.

The two tests shown, **GetUser**, and **AddUserFailure**, use different techniques to ensure that the underlying **Users.GetUser()** method is working correctly. In the **GetUser()** test, assertions are used to verify that all

properties of a user extracted from the database match up with expected values. Each **AssertEquals()** call succeeds if the second and third parameters are equal. If not, then the test fails and the message provided as the first parameter is displayed. The **AddUserFailure** test uses the **ExpectedException** attribute defined in the NUnit framework. In this case, the **AddUser** method should throw an exception if an attempt is made to add a user whose email already exists in the users database. This test will only succeed if an exception of the proper type is thrown. Most tests consist of some setup (anything not done in the global **SetUp** routine) followed by one or more assertions. In testing a data access layer, none of my tests have needed to be more than a few lines long (at most one assertion per column/property, as with the **GetUser()** test above). Once the tests are in place, the data access layer can be refactored with confidence. For example, in my production DAL code, I had the code from some time ago. I added the tests, made sure they covered everything I cared about, and then converted my methods to use the Microsoft® Data Access Application Block, greatly reducing the lines of code necessary. I reran the tests periodically to ensure that everything still worked. Once that was done, I implemented an intelligent caching layer within the DAL, which was largely transparent from the calling code and thus could be tested with the same set of tests (and a few new ones to test the caching features). Having the test suite allowed me to make these fairly drastic changes quickly and with confidence.

Performance

In order to maximize the performance of your test suite, you should make sure your setup and teardown scripts are doing as little as possible. Generally, the only test cases that matter with regard to quantity are 0, 1, and more than 1. Thus, there is generally little benefit derived from testing that a table works with fifty rows of data versus testing that it works with two or three rows.

If you have a subset of tests that require a substantial amount of data to be present, place these tests in their own

separate test fixture and create separate stored procedures for their setup and teardown.

Typically you'll

want these stored procedures to call **test_SetUp** and **test_TearDown** internally.

Upgrading to .NET 1.1

If you have NUnit version 2.0 and .NET 1.1, NUnit will not work by default. You can fix this by adding a <startup> section to NUnit's configuration file. Within the startup element, you'll specify

supportedRuntime and **requiredRuntime** versions, to force it to run under 1.1. You can do this for some

projects and not others if you want to continue running NUnit 2.0 under .NET 1.0 for some projects. The final section would look like this:

```
<startup>
<supportedRuntime version="v1.1.4322"/>
<requiredRuntime version="v1.1.4322"/>
</startup>
```

Note that the latest version of NUnit is 2.1 at the time of this writing, and provides support for .NET 1.1 intrinsically.

Configuration Files

Setting up configuration files for NUnit can be tricky if you haven't done it before. This is a very common requirement, especially in the data access scenario, since most data access layers depend on a configuration file for their database connection string information. In order to use a configuration file with NUnit, follow these steps:

1. Copy your ASP.NET web.config to your NUnit test project's output folder (for example, /bin).
2. Rename the copy to the same name as your NUnit test project assembly, with ".config" added to the end.

(if the assembly name is "AspAlliance.Data.UnitTests.dll" then the config file should be "AspAlliance.Data.UnitTests.dll.config").

3. Edit the new .config file and change your database settings so you are pointing to your test database, not production!

Sample Configuration File: **MyCompany.Security.Data.UnitTests.dll.config**

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<!-- Required for NUnit 2.0, but not for 2.1 -->
<!--
<startup>
<supportedRuntime version="v1.1.4322"/>
<requiredRuntime version="v1.1.4322"/>
</startup>
--> <appSettings>
<add key="MyApplication.ConnectionString"
value="server=localhost;
database=NUnit;Integrated Security=true" />
</appSettings>
</configuration>
```

Scale

Scalability is a valid concern with this approach. If we wanted to simulate production exactly, our setup procedure would backup the production database to the test database before each unit test. Assuming the database is more than a few megabytes, this would take far too long (unit tests should run in seconds, or at

worst a few minutes, but no longer, for all but the largest of applications). So, as long as only a sampling of production data is sufficient and covers the relevant variations in the production data, the setup procedure should only involve a few inserts per table, and thus should scale fairly well (see Performance, above).

At some point, for some applications, this technique will simply be too slow to use as frequently as TDD demands. At that point, one option is to scale back the frequency with which these tests are run. Darren Hobbs suggests this in his brief article, Putting the Unit in Unit Tests. Darren contends that most tests that rely on a lot of database setup are too slow and too brittle (prone to breaking during design changes) to be run all the time, as unit tests should be. I agree that if and when such tests do become a burden, one reasonable solution would be to only run them prior to checking code into production, rather than on every build or feature adjustment.

Another option worth exploring is to use mock objects instead of a test database. However, since we're concerned with testing the data access layer here, there is very little for the code to do without a database to talk to, which is why I haven't gone this route. Creating mock objects also takes more development time than writing a couple of setup and teardown stored procedures. A brief comparison of test databases vs. mock objects can be found from the Unit Testing Database Code on the DevDaily Web Site.

Summary

Using NUnit, you can quickly detect problems in your data access layer (before your users do). By following a test-first development style, you will write better code and have greater confidence in your ability to modify the code without breaking other parts of the application. Hopefully this article has made you curious to learn more about test driven development, NUnit, and how they can help you build high quality .NET applications.

Resources

- NUnit Web site
- NUnit Addin for Visual Studio.NET
- NUnitASP Web site
- TestDriven.com Web site
- XP Programming Web site
- Test Driven Development Group page on Yahoo
- Test Driven Development: By Example by Kent Beck—Book page on Amazon.com
- Refactoring: Improving the Design of Existing Code by Martin Fowler—Book page on Amazon.com

- .TEST from ParaSoft
- Unit Testing Database Code by Richard Dallaway
- Unit Testing Database Code on the DevDaily Web site

About the Author:

Steven A. Smith, Microsoft ASP.NET MVP, is president and owner of ASPAlliance.com. He is also the owner and head instructor for ASPSmith Ltd, a .NET-focused training company. He has authored two books, the ASP.NET Developer's Cookbook and ASP.NET By Example, as well as articles in MSDN® and AspNetPRO magazines. Steve speaks at several conferences each year and is a member of the INETA speaker's bureau. Steve has a Master's degree in Business Administration and a Bachelor of Science degree in Computer Science Engineering. Steve can be reached at ssmith@aspalliance.com.
http://www.ftponline.com/vsm/2003_09_14th/magazine/departments/firstlooks/page6.aspx

Automate .NET Assembly Unit Testing

by Ken Cox

.TEST 1.0 is a sophisticated automation tool for unit testing .NET assemblies. Its purpose is to allow you and

your manager to root out poorly written code and errors that could cause crashes. This automated errorprevention application takes you directly to the troublesome source code and even suggests a fix before the assembly gets off your hard drive.

You can point .TEST to a .NET assembly (DLL or EXE) and generate two types of reports. The static analysis shows violations of coding rules (drawn from Microsoft's Design Guidelines for Class Library Developers).

This feature evokes visions of a Dilbert cartoon where the boss uses .TEST's charts to show the number of times

Dilbert used "mysterious acronyms in property names." Fortunately, you can delete annoying rules, such as the

one that flags the parameter name "e" in VS.NET-generated code.

The dynamic analysis shows .TEST's real strength. It creates stubs (in C#, not VB.NET) and test cases that

exercise your functions with bad data, such as a Null where you expect a string. You can drill down into reports

that reveal the code's weaknesses and failures (see Figure 1). For example, you can see the test values, the

expected result, and the actual result for each call. You can edit these scenarios and retest for regression errors.

A valuable .TEST feature for Web developers would be a dynamic scan to ensure that user input is validated to thwart scripting and overflow attacks.

Parasoft boosts .TEST's performance by prescanning your .NET assemblies, but the tool seems sluggish, especially on startup, perhaps because it uses the Java runtime. The documentation is usable but could go further to teach developers what makes their code accident-prone. Installation of .TEST wasn't smooth. The VS.NET toolbar add-in failed, because .TEST wanted Visual C++ .NET—and the older 2002 version at that—installed on my machine. Parasoft support incorrectly diagnosed a failure to shut down VS.NET before the installation. (The setup should check for that condition anyway.) The final hurdle was a license key that disabled most product features. After all that, .TEST's VS.NET integration turned out to be somewhat shallow, consisting mainly of launching the standalone GUI and displaying troublesome source code. However, this valuable tool handles the mind-numbing drudgery of testing and retesting every line of code in every assembly in a project. If you're an independent developer, you might dream of automated bug hunting with .TEST at your workstation before a daily check-in. However, at .TEST's steep price—with the license tied to a single machine ID—a common build machine is a more likely installation location.

QUESTION NO: 40

You are an application developer for Testkingprep.in. Testkingprep maintains an internal, self-signed certification authority (CA). You are releasing a new internal Windows Forms application. A written company policy prohibits internal applications from running on client computers unless the identity and integrity of those applications can be proven. The Microsoft .NET Framework on all client computers is configured to enforce this restriction.

You need to ensure that your application will run when installed on all client computers. Your solution must not require any financial expenditure.

What should you do?

- A. Use a software publisher certificate issued by the internal CA to sign the application assemblies.
- B. Use a software publisher certificate issued by a third-party commercial CA to sign the application assemblies.
- C. Run the Certificate Creation tool and the Software Publisher Certificate Test tool before distributing the application to client computers.
- D. Distribute the application as an e-mail attachment. Digitally sign the e-mail message before sending it to all company users.

Answer: A

Explanation

Getting a Software Publisher Certificate

Before you can sign files, you need to obtain a Software Publisher Certificate (SPC). To do this, you must make

a request to a Certification Authority. During the application process, you must generate a key pair and provide

the Certification Authority with identification information, such as your name, address, and public key. You

must also make a legally binding pledge that you cannot and will not distribute software you know or should

have known contains viruses or will otherwise maliciously harm the user's machine or code.

For more information about obtaining a Software Publisher Certificate (SPC), see

<http://www.microsoft.com/intdev/security/authcode/codesign.htm>. To apply for a certificate, see

<http://www.microsoft.com/intdev/security/authcode/certs.htm>. To create a test certificate to test signing your

files, see Making a Test Software Publisher Certificate.

The Certification Authority generates a Software Publisher Certificate that conforms to the industry standard

X.509 certificate format with Version 3 extensions. The certificate identifies you and contains your public key.

It is stored by the Certification Authority for reference and a copy is returned to you via electronic mail. After

accepting the certificate, you should include a copy in all published software signed with the private key.

Software Publisher Certificate Test Tool (Cert2spc.exe)

The Software Publisher Certificate Test tool creates a Software Publisher's Certificate (SPC) from one or more

X.509 certificates. Cert2spc.exe is for test purposes only. You can obtain a valid SPC from a Certification

Authority such as VeriSign or Thawte. For more information on creating X.509 certificates, see the Certificate

Creation Tool (Makecert.exe).

cert2spc *cert1.cer [cert2.cer ... certN.cer] outputSPCfile.spc*

Argument Description

certN.cer The name of an X.509 certificate to include in the SPC file. You can specify multiple

names separated by spaces.

outputSPCfile.spc The name of the PKCS #7 object that will contain the X.509 certificates. You can use

the .spc file as input to the File Signing Tool (Signcode.exe).

Option Description

/? Displays the command syntax for the tool.

Examples

The following command creates an SPC from myCertificate.cer and places it in mySPCFile.spc.

```
cert2spc myCertificate.cer mySPCFile.spc
```

The following command creates an SPC from oneCertificate.cer and twoCertificate.cer and places it in mySPCFile.spc.

```
cert2spc oneCertificate.cer twoCertificate.cer mySPCFile.spc
```

File Signing Tool (Signcode.exe)

The File Signing tool signs a portable executable (PE) file (.dll or .exe file) with an Authenticode digital

signature. You can sign either an assembly or an individual file contained in a multifile assembly. If you are

distributing an assembly, you should sign the assembly rather than the individual files.

Running Signcode.exe

without specifying any options launches a wizard that helps with signing.

signcode [*options*] *filename* | *assemblyname*

Argument Description

filename The name of the PE file to sign.

assemblyname The name of the assembly to sign. This file must contain an assembly manifest.

Option Description

-\$ authority Specifies the signing authority of the certificate, which must be either **individual** or

commercial. By default, Signcode.exe uses the certificate's highest permission.

-a algorithm Specifies the hashing algorithm for signing, which must be either **md5** (the default) or

sha1.

-c file Specifies the file that contains the encoded software publishing certificate.

-cn name Specifies the common name of the certificate.

-i info Specifies a place to get more information on content (usually a URL).

-j dllName Specifies the name of a DLL that returns an array of authenticated attributes for signing

files. You can specify more than one DLL by repeating the **-j** option.

-jp param Specifies a parameter to be passed for the preceding DLL. For example: **-j dll1**

-jp

dll1Param. The tool allows only one parameter per DLL.

-k keyname Specifies the key container name.

-ky keytype Specifies the key type, which must be **signature**, **exchange**, or an integer (such as 4).

-n name Specifies a text name that represents the content of the file to sign.

-p provider Specifies the name of the cryptographic provider on the system.

-r location Specifies the location of the certificate store in the registry, which must be either

currentuser (the default) or **localmachine**.

-s store Specifies the certificate store that contains the signing certificate. The default is **my** store.

-sha1 thumbprint Specifies the *thumbprint*, which is the sha1 hash of the signing certificate included in the certificate store.

-sp policy Sets the certificate store policy, which must be either **spcStore** (the default) or **chain**.

If you specify **chain**, all certificates in the verification chain, including self-signed certificates, are added to the signature. If you specify **spcStore**, trusted, self-signed certificates are not included with the certificates in the chain that are added to the signature.

-spc file Specifies the SPC *file* that contains software publishing certificates.

-t URL Indicates that the file is to be timestamped by the timestamp server at the specified http address.

-tr number Specifies the maximum number of timestamp trials until success; defaults to 1.

-tw number Specifies the delay (in number of seconds) between each timestamp trial. Defaults to 0.

-v pvkFile Specifies the private key (.pvk) file name that contains the private key.

-x Timestamps, but does not sign, the file.

-y type Specifies the cryptographic provider type to use.

-? Displays command syntax and options for the tool.

Remarks

To sign with a software publisher certificate (SPC) file, you must specify the **-spc** and **-v** options if your private key is in a PVK file. If your private key is in a registry key container, you must specify the **-spc** and **-k** options.

If you want to sign your file with an SPC file, you should create the SPC file using the Certificate Creation tool and the Software Publisher Certificate Test tool.

Examples

The following command signs XYZ.exe using the XYZ.spc Software Publisher Certificate and the private key in the registry key container XYZ.

```
signcode /spc XYZ.spc /k XYZ XYZ.exe
```

The following command signs the assembly myAssembly using the certificate in myCertificate.spc

and the private key in myKey.pvk.

```
signcode /spc myCertificate.spc /v myKey.pvk myAssembly
```

Certificate Creation Tool (Makecert.exe)

The Certificate Creation tool generates X.509 certificates for testing purposes only. It creates a public and private key pair for digital signatures and stores it in a certificate file. This tool also associates the key pair with

a specified publisher's name and creates an X.509 certificate that binds a user-specified name to the public part of the key pair.

Makecert.exe includes basic and extended options. Basic options are those most commonly used to create a certificate. Extended options provide more flexibility.

makecert [*options*] *outputCertificateFile*

Argument Description

outputCertificateFile The name of the .cer file where the test X.509 certificate will be written.

Basic Options

Option Description

-n *x509name* Specifies the subject's certificate name. This name must conform to the X.500 standard.

The simplest method is to specify the name in double quotes, preceded by **CN=**; for example, "CN=*myName*".

-sk *keyname* Specifies the subject's key container location, which contains the private key. If a key container does not exist, it will be created.

-sr *location* Specifies the subject's certificate store location. *Location* can be either **currentuser** (the default), or **localmachine**.

-ss *store* Specifies the subject's certificate store name that stores the output certificate.

-# *number* Specifies a serial Number from 1 to 2³¹-1. The default is a unique value generated by Makecert.exe.

-\$ *authority* Specifies the signing authority of the certificate, which must be set to either **commercial**

(for certificates used by commercial software publishers) or **individual** (for certificates used by individual software publishers).

-? Displays command syntax and a list of basic options for the tool.

-! Displays command syntax and a list of extended options for the tool.

Extended Options

Option Description

-a *algorithm* Specifies the signature algorithm. Must be either **md5** (the default) or **sha1**.

-b *mm/dd/yyyy* Specifies the start of the validity period. Defaults to the certificate's creation date.

-cy *certType* Specifies the certificate type. Valid values are **end** for end-entity, **authority** for certification authority, or **both**.

-d *name* Displays the subject's name.

-e *mm/dd/yyyy* Specifies the end of the validity period. Defaults to 12/31/2039 11:59:59 GMT.

-eku *oid[,oid]* Inserts a list of comma-separated, enhanced key usage object identifiers (OIDs) into the certificate.

- h number** Specifies the maximum height of the tree below this certificate.
- ic file** Specifies the issuer's certificate file.
- ik keyName** Specifies the issuer's key container name.
- iky keytype** Specifies the issuer's key type, which must be **signature**, **exchange**, or an integer (such as 4).
- in name** Specifies the issuer's certificate common name.
- ip provider** Specifies the issuer's CryptoAPI provider name.
- ir location** Specifies the location of the issuer's certificate store. *Location* can be either **currentuser** (the default) or **localmachine**.
- is store** Specifies the issuer's certificate store name.
- iv pvkFile** Specifies the issuer's .pvk private key file.
- iy pvkFile** Specifies the issuer's CryptoAPI provider type.
- l link** Links to policy information (for example, a URL).
- m number** Specifies the duration, in months, of the certificate validity period.
- nscp** Includes the Netscape client-authorization extension.
- r** Creates a self-signed certificate.
- sc file** Specifies the subject's certificate file.
- sky keytype** Specifies the subject's key type, which must be **signature**, **exchange**, or an integer (such as 4).
- sp provider** Specifies the subject's CryptoAPI provider name.
- sv pvkFile** Specifies the subject's .pvk private key file. The file is created if none exists.
- sy type** Specifies the subject's CryptoAPI provider type.

Examples

The following command creates a test certificate and writes it to testCert.cer.

```
makecert testCert.cer
```

The following command creates a test certificate and writes it to testXYZ.cer, using the subject's key container and the certificate subject's X.500 name.

```
makecert -sk XYZ -n "CN=XYZ Company" testXYZ.cer
```

QUESTION NO: 41

You are an application developer for Testkingprep.in. You are developing an ASP.NET Web application on your Microsoft Windows XP Professional client computer. Your computer has IIS 5.1 installed and is hosting the development project. Your computer also has the most recent version of the Microsoft .NET Framework installed. The completed application will run under IIS 6.0.

You need to ensure that your testing accurately reflect the security configuration of the production environment.

What should you do before testing the application?

- A. Install the .NET Framework 1.0 on the production Web server.
- B. Deploy the application to a Microsoft Windows Server 2003 test computers.
- C. Install the most recent security updates for Microsoft Visual Studio .NET 2003 on your client computer.

D. Disable the ASPNET user account on your client computer.

Answer: B

Explanation

It is an industry best practice to never place untested code in a live environment. The best practice is to setup a lab environment that mimics the production and perform testing there prior to deployment. It is easier to rollback a test environment than the production environment. In most cases a rollback to a pristine test environment is mandatory to ensure the subsequent testing is not impacted by previous test with different applications. Since IIS6 can be configured to be much more restrictive than earlier versions, this validation is required.

Software Testing Best Practices

<http://www.chillarege.com/authwork/papers1990s/TestingBestPractice.pdf>

Desktop Deployment Web Seminar

[http://www.microsoft.com/downloads/details.aspx?FamilyID=2bc24b8c-ba04-45c3-bcd0-](http://www.microsoft.com/downloads/details.aspx?FamilyID=2bc24b8c-ba04-45c3-bcd0-12af0e718e79&DisplayLang=en)

[12af0e718e79&DisplayLang=en](http://www.microsoft.com/downloads/details.aspx?FamilyID=2bc24b8c-ba04-45c3-bcd0-12af0e718e79&DisplayLang=en)

QUESTION NO: 42

You are an application developer for Testkingprep.in. Testkingprep sells products to business partners over the Internet. Partners sometimes claim that they did not place an order that Testkingprep fulfilled.

You need to develop a solution that can be used to prove that a partner's order were approved by that partner. You want your solution to be as secure as possible.

What should you do?

A. Require each partner to identify itself by using an account name.

B. Require all orders to be placed at an HTTPS Web site that uses Testkingprep's certificate.

C. Require partners to digitally sign each order by using a certificate.

D. Require partners to send an e-mail message to confirm each order.

Answer: C

Explanation

A digital signature

A means for originators of a message, file, or other digitally encoded information to bind their identity to the

information. The process of digitally signing information entails transforming the information, as well as some

secret information held by the sender, into a tag called a *signature*. Digital signatures are used in public key

environments, and they provide nonrepudiation and integrity services.

It is a way to ensure the integrity and origin of data. A digital signature provides strong evidence that the data

has not been altered since it was signed and it confirms the identity of the person or entity who signed the data.

This enables the important security features of *integrity* and *nonrepudiation*, which are essential for secure electronic commerce transactions.

Digital signatures are typically used when data is distributed in plaintext, or unencrypted form. In these cases,

while the sensitivity of the message itself may not warrant encryption, there could be a compelling reason to

ensure that the data is in its original form and has not been sent by an impostor because, in a distributed

computing environment, plaintext can conceivably be read or altered by anyone on the network with the proper access, whether authorized or not.

nonrepudiation: 1. The capability, in security systems, that guarantees that a message or data can be proven to

have originated from a specific person. 2. Assurance the sender of data is provided with proof of delivery and

the recipient is provided with proof of the sender's identity, so neither can later deny having processed the data.

Nonrepudiation

Nonrepudiation is a method of proving either that a user performed an action (such as enrolling in a stock plan

or applying for a car loan), or that the user sent or received some information at a particular time. This prevents

the individual from fraudulently reneging on a transaction. By comparison, if you purchase an item, you might

have to sign for the item upon receipt. If you decide to renege on the deal, the vendor can simply show you the

signed receipt.

A comprehensive nonrepudiation plan usually requires authentication, authorization, data integrity, and

auditing. In addition, nonrepudiation requires a message on the Web page, warning that the action the user is about to take is legally binding. This does not make the Web server more secure, but it does make Web transactions (such as purchasing an item) more secure.

Today, electronic nonrepudiation across the Internet is new and there is little in the way of legal precedent. This is sure to change as more business is performed across the Web.

QUESTION NO: 43

You are an application developer for Testkingprep.in, which is a financial services company. You are developing an ASP.NET Web application that will be used by Testkingprep's customers. Customers will use the application to access their portfolios and to view business and financial reports. The customers are divided into two categories named Standard and Premier. The Premier customers will have access to an additional set of reports and analysis. You

plan to use roles named Standard and Premier to differentiate the two customer categories.

The application will use Forms authentication to authenticate all users and assign each authenticated user to either the Standard role or the Premier role. Web pages that are accessible only by Premier customers are in a subfolder named Premier.

Web pages that are accessible by both categories of customers are in the application root.

You need to configure URL authorization for the application. You plan to achieve this goal by adding configuration elements to the Web.config file in the application root.

Which elements should you use?

A. <authorization>
<deny users=?"/>
</authorization>
<location path="Premier">
<system.web>
<authorization>
<allow roles="Premier"/>
<deny users="*/>
</authorization>
</system.web>
</location>

B. <authorization>
<deny users="?"/>
</authorization>
<location path="Premier">
<system.web>
<authorization>
<deny users="*/>
<allow roles="Premier"/>
</authorization>
</system.web>
</location>

C. <authorization>
<deny users="?"/>
<deny roles="Premier"/>
<allow users="*/>
</authorization>
<location path="Premier">
<system.web>
<authorization>
<allow roles="Premier"/>
</authorization>
</system.web>
</location>

```
D. <authorization>
<deny users="?">/>
</authorization>
<location path="Premier">
<system.web>
```

Answer: A

Explanation

URL Authorization

Internet Information Services (IIS) 6.0 works with Authorization Manager, a management tool that is available with the Microsoft® Windows® Server 2003 family of operating systems, to implement IIS URL authorization.

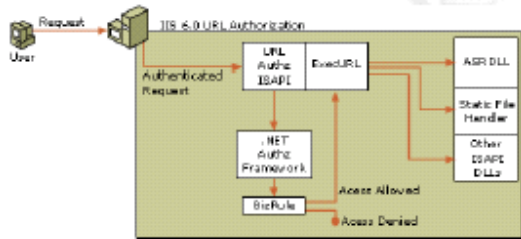
Overview

Authorizing user access to Web application resources requires the management of many Access Control Lists (ACLs). In turn, maintaining ACLs requires administrators to track precisely which permissions are needed on each resource for each user or group to perform meaningful tasks. IIS URL authorization allows Windows administrators to simplify access management by authorizing user access to the URLs that comprise a Web application.

When a user requests access to a URL, IIS URL authorization validates the user's access based on that user's roles, which can be defined in Lightweight Directory Access Protocol (LDAP) queries, custom user roles, and Authorization Manager scripts (BizRules). This allows administrators to simplify access control management by controlling all user access to URLs instead of controlling access per ACL on each resource.

IIS URL authorization is implemented as an Internet Server API (ISAPI) interceptor (in the diagram below, URL Authz ISAPI). When an application, virtual directory, or URL is configured to use IIS URL authorization, each request to a URL will be routed to the URL authorization ISAPI interceptor. The URL authorization ISAPI interceptor will use Authorization Manager (in the diagram, .NET Authz Framework) to authorize access to the requested URL. The URL must be associated with an Authorization Manager policy store that contains the authorization policy for the URL. Once the client has been authorized to access the URL, the URL authorization

ISAPI's Execute URL feature (in the diagram, ExecURL) will pass the request to the appropriate handler for the URL, such as ASP.dll, another ISAPI, or the Static File Handler.



By using IIS 6.0 URL authorization, an administrator can control access based on information that is only available at runtime. For example, if you have a Web page that should only be available to employees in a given cost center or to employees of a certain age, you can assign roles to the correct users based on LDAP queries that will check the cost center or age attributes on a user's object. If employees can only access certain pages on certain days of the week or during a certain time of day, a BizRule can be created which grants access to the URL based on these values or any value that can be asserted at runtime, including IIS Server Variables.

Using URL Authorization

To use URL authorization in IIS 6.0 you must enable the ISAPI interceptor, Urlauth.dll. In addition, you must set the following metabase properties on the application, virtual directory, or URL (Web site):

1. **AzEnable**: Enables URL authorization for the virtual directory, application, or URL that corresponds to the entry in the metabase.
2. **AzStoreName**: Associates an Authorization Manager store with the virtual directory, application, or URL.
3. **AzScopeName**: Associates the virtual directory, application, or URL with a scope. This scope will be the name of a scope in the IIS 6.0 URL authorization application in the Authorization Manager policy store referred to in the **AzStoreName** attribute. If no scope or an empty string is specified, the default scope of the IIS 6.0 URL authorization will be used.
4. **AzImpersonationLevel**: Determines the impersonation behavior for the application. This allows you to configure the Web application to impersonate the client user, the IIS worker process, or the IUSER_computername account for the worker process. Each setting significantly changes the environment and implied design of the Web application.

Sample Script

The sample script below, written in Microsoft Visual Basic® Scripting Edition (VBScript), marks the root of the first site as a URL in "MyAZScope", which is defined in the MyAZStore.xml file.

Users with URLAccess

rights in this scope will be able to access the site.

```
var objVDir = GetObject("IIS://localhost/w3svc/1/root");  
objVDir.AzEnable = true;  
objVDir.AZStoreName = "MSXML://d:\MyAZStore.xml";  
objVDir.AzScopeName = "MyAZScope";  
objVDir.AZImpersonationLevel = 0;  
objVDir.SetInfo();
```

While URL authorization controls access to other forms of authorization, such as ACLs or IIS directory security

permissions settings, the application context still requires the correct IIS directory security and ACL

permissions. IIS URL authorization allows the IIS directory security and ACL permissions to be more easily maintained.

When IIS 6.0 URL authorization is configured, the **AzStoreName** attribute in the IIS metabase entry for the

application, virtual directory, or URL will identify an Authorization Manager policy store. To manage the

authorization policy, run Authorization Manager and use the Open Policy Store. IIS 6.0 URL authorization is an

application in this store. The **AzScopeName** attribute in the metabase entry will be an authorization manager

scope in the IIS 6.0 URL authorization application. Use this scope to manage access to the corresponding URL.

When configuring an application, virtual directory, or URL for URL authorization, a scope must be created in

the authorization policy store with the same name as that specified in the corresponding metabase entries

AzScopeName attribute.

Enabling the ISAPI Interceptor

To use the URL authorization ISAPI interceptor (Urlauth.dll), you must first enable it for each Web site that

requires URL authorization.

Important You must be a member of the Administrators group on the local computer to perform the following

procedure (or procedures), or you must have been delegated the appropriate authority. As a security best

practice, log on to your computer using an account that is not in the Administrators group, and then use the Run

as command to run IIS Manager as an administrator. From the command prompt, type **runas**

/user:administrative_accountname "mmc %systemroot%\system32\inetsrv\iis.msc".

To enable the URL authorization ISAPI interceptor

1. In IIS Manager, expand the local computer, expand the **Web Sites** folder, right-click the Web site that you want, and then click **Properties**.
2. Click the **Home Directory** tab, and then in the **Application settings** section, click **Configuration**.
3. Click the **Mappings** tab, and then in the **Wildcard application maps** section, click **Insert**.
4. In the **Add/Edit Application Extension Mapping** box, click **Browse** and browse to the **Windows\system32\inetsrv** directory.
5. Click **urlauth.dll**, click **Open**, and then click **OK**.

Related Topics

- For more information on Authorization Manager, see Authorization Manager in Windows Help

<authorization> Element

Configures ASP.NET authorization support. The **<authorization>** tag helps control client access to URL

resources. This element can be declared at any level (machine, site, application, subdirectory, or page).

<configuration>

<system.web>

<authorization>

<authorization>

<allow users="comma-separated list of users"

roles="comma-separated list of roles"

verbs="comma-separated list of verbs"/>

<deny users="comma-separated list of users"

roles="comma-separated list of roles"

verbs="comma-separated list of verbs"/>

</authorization>

Subtags

Subtag Description

<allow> Allows access to a resource based on the following:

users: A comma-separated list of user names that are granted access to the resource. A question mark

(?) allows anonymous users; an asterisk (*) allows all users.

roles: A comma-separated list of roles that are granted access to the resource.

verbs: A comma-separated list of HTTP transmission methods that are granted access to the resource.

Verbs registered to ASP.NET are GET, HEAD, POST, and DEBUG.

<deny> Denies access to a resource based on the following:

users: A comma-separated list of user names that are denied access to the resource. A question mark

(?) indicates that anonymous users are denied access; an asterisk (*) indicates that all users are denied access.

roles: A comma-separated list of roles that are denied access to the resource.

Subtag Description

verbs: A comma-separated list of HTTP transmission methods that are denied access to the resource.

Verbs registered to ASP.NET are GET, HEAD, POST, and DEBUG.

Remarks

At run time, the authorization module iterates through the **<allow>** and **<deny>** tags until it finds the first

access rule that fits a particular user. It then grants or denies access to a URL resource depending on whether

the first access rule found is an **<allow>** or a **<deny>** rule. The default authorization rule in the Machine.config

file is **<allow users="*" />** so, by default, access is allowed unless configured otherwise.

Top of page

Example

The following example allows access to all members of the Admins role and denies access to all users.

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

QUESTION NO: 44

You are an application developer for Testkingprep.in. You develop an application that receives data from a remote component.

You are developing a method to detect any corrupted incoming data and log information to a file for analysis. You plan to use two functions. A function named PrepareTestkingprepData will be called by the remote component. The second function will be called by the local application to verify that the data was not corrupted during transmission.

You need to ensure that corrupted data can be identified.

Which code segment should you use?

```
A. static byte[] PrepareTestkingprepData(byte[] data) (
MemoryStream ms=new MemoryStream();
ms.Write(data,0,data.Length);
ms.Write(data,0,data.Length);
return ms.ToArray();
```

```

B. static byte[] PrepareTestkingprepData(byte[] data) (
MD5 md5=new MD5CryptoServiceProvider();
MemoryStream ms=new MemoryStream();
ms.Write(md5.ComputeHash(data), 0, md5.HashSize);
ms.Write(data,0,data.Length);
return ms.ToArray();
)
C. static byte[] Preparation(byte[] data) (
DES des=new DESCryptoServiceProvider();
MemoryStream ms=new MemoryStream();
ms.Write(des.Key,0,des.Key.Length);
ms.Write(des.IV,0,des.IV.Length);
CryptoStream cs=new CryptoStream(ms, des.CreateEncryptor(), CryptoSt
cs.Write(data,0,data.Length);
cs.FlushFinalBlock();
return ms.ToArray();
)
D. static byte[] PrepareTestkingprepData(byte[] data) (
MemoryStream ms=new MemoryStream();
StreamWriter sw=new StreamWriter(ms,Encoding.UTF8);
sw.Write(Encoding.UTF8.GetString(data));
return ms.ToArray();

```

Answer: B

Explanation

Hash functions map binary strings of an arbitrary length to small binary strings of a fixed length. A

cryptographic hash function has the property that it is computationally infeasible to find two distinct inputs that

hash to the same value; that is, hashes of two sets of data should match if the corresponding data also matches.

Small changes to the data result in large unpredictable changes in the hash.

Example

The following example computes the **MD5** hash for data and stores it in result. This example assumes that

there is a predefined constant DATA_SIZE.

```
Dim data(DATA_SIZE) As Byte
```

' This is one implementation of the abstract class MD5.

```
Dim md5 As New MD5CryptoServiceProvider()
```

```
Dim result As Byte() = md5.ComputeHash(data)
```

It is easy to generate and compare hash values using the cryptographic resources contained in the

System.Security.Cryptography namespace. Because all hash functions take input of type **Byte[]**, it might be

necessary to convert the source into a byte array before it is hashed. To create a hash for a string value, follow these steps:

1. Open Visual Studio .NET.
2. Create a new Console Application in Visual Basic .NET. Visual Studio .NET creates a Module for you along with an empty Main() procedure.
3. Make sure that the project references the **System** and **System.Security** namespaces.
4. Use the **Imports** statement on the **System**, **System.Security**, **System.Security.Cryptography**, and **System.Text** namespaces so that you are not required to qualify declarations from these namespaces later in your code. These statements must be used prior to any other declarations.
5. Imports System
6. Imports System.Security
7. Imports System.Security.Cryptography
8. Imports System.Text
9. Declare a string variable to hold your source data, and two byte arrays (of undefined size) to hold the source bytes and the resulting hash value.
10. Dim sSourceData As String
11. Dim tmpSource() As Byte
12. Dim tmpHash() As Byte
13. Use the **GetBytes()** function, which is part of the **System.Text.ASCIIEncoding.ASCII** class, to convert your source string into an array of bytes (required as input to the hashing function).
14. sSourceData = "MySourceData"
15. 'Create a byte array from source data.
16. tmpSource = ASCIIEncoding.ASCII.GetBytes(sSourceData)
17. Compute the MD5 hash for your source data by calling **ComputeHash** on an instance of the **MD5CryptoServiceProvider** class. Note that to compute another hash value, you will need to create another instance of the class.
18. 'Compute hash based on source data.
19. tmpHash = New MD5CryptoServiceProvider().ComputeHash(tmpSource)
20. The tmpHash byte array now holds the computed hash value (128-bit value=16 bytes) for your source data. It is often useful to display or store a value like this as a hexadecimal string, which the following code accomplishes:
21. Console.WriteLine(ByteArrayToString(tmpHash))
- 22.
23. Private Function ByteArrayToString(ByVal arrInput() As Byte) As String
24. Dim i As Integer
25. Dim sOutput As New StringBuilder(arrInput.Length)

26. For i = 0 To arrInput.Length - 1
27. sOutput.Append(arrInput(i).ToString("X2"))
28. Next
29. Return sOutput.ToString()
30. End Function
31. Save and then run your code to see the resulting hexadecimal string for the source value.

QUESTION NO: 45

You are an application developer for Testkingprep.in. You develop a library assembly that contains diagnostic utility classes. This library assembly is installed in the global assembly cache on all client computers on Testkingprep's network. You develop a Windows Forms application that calls the library assembly. You successfully test the application on your computer, and then you deploy the application to a Web folder on the intranet.

Further testing reveals that when you run this application from the intranet, a SecurityException exception is thrown when the application is loading. You need to correct the problem that is causing the SecurityException exception. What should you do?

- A. Add the following code segment to the library assembly.
[assembly: AllowPartialTrustedCallers]
- B. Add the following code segment to the Windows Forms application assembly.
[assembly: AllowPartiallyTrustedCallers]
- C. Add the following code segment to the library assembly.
[assembly: PermissionSet(SecurityAction.RequestOptional, Name = "LocalIntranet")]
- D. Add the following code segment to the Windows Forms application assembly.
[assembly: PermissionSet(SecurityAction.RequestMinimum, Name = "LocalIntranet")]

Answer: D

Explanation

.NET permissions are grouped into NamedPermissionSets. The platform includes the following non-modifiable built-in sets: Nothing, Execution, FullTrust, Internet, LocalIntranet, SkipVerification. The FullTrust set is a special case, as it declares that this code does not have any restrictions and passes any permission check, even for custom permissions. By default, all local code (found in the local computer directories) is granted this privilege.

The above fixed permission sets can be demanded instead of regular permissions:

```
[assembly:PermissionSetAttribute(  
SecurityAction.RequestMinimum,  
Name="LocalIntranet")]
```

Here is a summary of some facts or rules:

7. If you want to restrict the permissions given to an assembly to only those contained in the associated

permission set, you must tick the code group option “The policy level will only have the permissions from the permission set associated with this code group”. Otherwise what is granted to the assembly is the permissions of the particular associated permission set plus permissions of the associated permission set of the inherited code group (“All_Code” group).

8. All assemblies must be given “Enable assembly execution” security permission so that it can be run or launched.

9. Permissions included in an assembly’s associated permission set that are above the logged-in user’s privilege will not be granted.

10. **A strongly named assembly can only be called by a fully-trusted caller, unless this assembly**

states AllowPartiallyTrustedCallers. When you use this attribute, it means that you have fully reviewed your code and there is no security flaw that may be used by luring attackers – such as a improperly used

Assert. Not all system assemblies are marked with this attribute. You can look at the assembly’s manifest to see whether it has that attribute.

11. However, an assembly belonging to the root “All_Code” code group can be called by partially-trusted

callers, even if they are strongly named. This is probably because, if you don’t impose a particular

security control on an assembly, the runtime security thinks that this assembly is not extremely critical.

12. When you states AllowPartiallyTrustedCallers in an assembly, or let it stay in the “All_Code” code group, a permission-checking stack walk is still going to be triggered for every attempt to access any controlled resource. The only difference is if you improperly make a Assert you will make luring attacks possible.

QUESTION NO: 46

You are an application developer for Testkingprep.in. You are developing an application that needs to exchange a shared key to start of each communication with remote components. The exchange occurs over the Internet.

You need to ensure that only the intended recipient can read the shared key.

What should you do?

- A. Sign the shared key by using the application’s private key.
- B. Encrypt the shared key by using the application’s private key.
- C. Encrypt the shared key by using the remote component’s public key.
- D. Encode the shared key by using the System.Text.Encoding.UTF8 object.
- E. Encode the shared key by using the System.Text.Encoding.BigEndianUnicode object.

Answer: C

Explanation

public key

A cryptographic key typically used when decrypting a session key or a digital signature. The public key can also be used to encrypt a message, guaranteeing that only the person with the corresponding private key can decrypt the message.

private key

The secret half of a key pair used in a public key algorithm. Private keys are typically used to encrypt a symmetric session key, digitally sign a message, or decrypt a message that has been encrypted with the corresponding public key.

Public/Private Key Pairs

Public/private key pairs are used for *asymmetric encryption*. Asymmetric encryption is used mainly to encrypt and decrypt *session keys* and *digital signatures*. Asymmetric encryption uses *public key encryption* algorithms.

Public key algorithms use two different keys: a *public key* and a *private key*. The private key member of the pair must be kept private and secure. The public key, however, can be distributed to anyone who requests it. The public key of a key pair is often distributed by means of a *digital certificate*. When one key of a key pair is used to encrypt a message, the other key from that pair is required to decrypt the message. Thus if user A's public key is used to encrypt data, only user A (or someone who has access to user A's private key) can decrypt the data. If user A's private key is used to encrypt a piece of data, only user A's public key will decrypt the data, thus indicating that user A (or someone with access to user A's private key) did the encryption.

If the private key is used to sign a message, the public key from that pair must be used to validate the signature.

For example, if Alice wants to send someone a digitally signed message, she would sign the message with her

private key, and the other person could verify her signature by using her public key.

Because presumably only

Alice has access to her private key, the fact that the signature can be verified with Alice's public key indicates

that Alice created the signature.

Unfortunately, public key algorithms are very slow, roughly 1,000 times slower than symmetric algorithms. It is

impractical to use them to encrypt large amounts of data. In practice, public key algorithms are used to encrypt session keys. Symmetric algorithms are used for encryption/decryption of most data. Similarly, because signing a message in effect encrypts the message, it is not practical to use public key signature algorithms to sign large messages. Instead, a fixed-length *hash* is made of the message and the hash value is signed. For more details, see Hashes and Digital Signatures. Each user generally has two public/private key pairs. One key pair is used to encrypt session keys and the other to create digital signatures. These are known as the *key exchange key pair* and the *signature key pair*, respectively.

Note that although key containers created by most *cryptographic service providers* (CSPs) contain two key pairs, this is not required. Some CSPs do not store any *key pairs* while other CSPs store more than two pairs. All keys in CryptoAPI are stored within CSPs. CSPs are also responsible for creating the keys, destroying them, and using them to perform a variety of cryptographic operations. Exporting keys out of the CSP so that they can be sent to other users is discussed in Cryptographic Key Storage and Exchange.

QUESTION NO: 47

You are an application developer for Testkingprep.in. You are developing a Windows Forms scheduling application for a medical clinic. Each user of the application belongs to either a Windows group named Receptionist or a Windows group named Manager. The domain name is Testkingprep.

The business rules of the application state that receptionists and managers can add and remove appointments to time slots in the schedule. However, only managers can mark time slots as unavailable for scheduled appointments. The application includes the following method, which is used to mark time slots as unavailable.

private void BlockOutTime(int physicianID, DateTime dtStart, DateTime dtEnd)

You need to modify the code to ensure that an exception is thrown when the BlockOutTime method is called by a user who is not a member of the Manager group.

What should you do?

- A. Add the following attribute to the method.
[PrincipalPermission(SecurityAction.Demand, Role=@"Testkingprep\Manager
- B. Add the following attribute to the method
[SecurityRole("Manager")]
- C. Add the following attribute to the method.
[SecureMethod]
- D. Add the following code segment to the beginning of the method.

```
PrincipalPermission p = new PrincipalPermission(null, @"Testkingprep\Ma  
p.IsUnrestricted();
```

Answer: A

Explanation

Assembly Design Considerations

One of the most significant issues to consider at design time is the trust level of your assembly's target environment, which affects the code access security permissions granted to your code and to the code that calls your code. This is determined by code access security policy defined by the administrator, and it affects the types of resources your code is allowed to access and other privileged operations it can perform.

When designing your assembly, you should:

- **Identify privileged code**
- **Identify the trust level of your target environment**
- **Sandbox highly privileged code**
- **Design your public interface**

Identify Privileged Code

Identify code that accesses secured resources or performs security sensitive operations.

This type of code

requires specific code access security permissions to function.

Identify Privileged Resources

Identify the types of resources your assembly needs to access; this allows you to identify any potential problems

that are likely to occur if the environment your assembly ultimately runs in does not grant the relevant code

access security permissions. In this case you are forced either to update code access security policy for your

application if the administrator allows this, or you must sandbox your privileged code.

For more information

about sandboxing, see Chapter 9, "Using Code Access Security with ASP.NET."

Identify Privileged Operations

Also identify any privileged operations that your assembly needs to perform, again so that you know which

code access permissions your code requires at runtime.

Identify the Trust Level of Your Target Environment

The target environment that your assembly is installed in is important because code access security policy may

constrain what your assembly is allowed to do. If, for example, your assembly depends on the use of OLE DB,

it will fail in anything less than a full trust environment.

Full Trust Environments

Full trust means that code has an unrestricted set of code access security permissions, which allows the code to access all resource types and perform privileged operations, subject to operating system security. A full trust environment is the default environment for a Web application and supporting assemblies installed on a Web server, although this can be altered by configuring the **<trust>** element of the application.

Partial Trust Environment

A partial trust environment is anything less than full trust. The .NET Framework has several predefined trust levels that you can use directly or customize to meet your specific security requirements. The trust level may also be diminished by the origin of the code. For example, code on a network share is trusted less than code on the local computer and as a result is limited in its ability to perform privileged operations.

Supporting Partial Trust Callers

The risk of a security compromise increases significantly if your assembly supports partial trust callers (that is, code that you do not fully trust.) Code access security has additional safeguards to help mitigate the risk. For additional guidelines that apply to assemblies that support partial trust callers, see Chapter 8, "Code Access Security in Practice." Without additional programming, your code supports partial trust callers in the following two situations:

- Your assembly does not have a strong name.
- Your assembly has a strong name and includes the **AllowPartiallyTrustedCallersAttribute** (APTCA) assembly level attribute.

Why Worry About the Target Environment?

The trust environment that your assembly runs in is important for the following reasons:

- A partial trust assembly can only gain access to a restricted set of resources and perform a restricted set of operations, depending upon which code access security permissions it is granted by code access security policy.

- A partial trust assembly cannot call a strong named assembly unless it includes **AllowPartiallyTrustedCallersAttribute**.

- Other partial trust assemblies may not be able to call your assembly because they do not have the necessary permissions. The permissions that a calling assembly must be able to call your assembly are determined by:

- The types of resources your assembly accesses
- The types of privileged operation your assembly performs

Sandbox Highly Privileged Code

To avoid granting powerful permissions to a whole application just to satisfy the needs of a few methods that

perform privileged operations, sandbox privileged code and put it in a separate assembly.

This allows an

administrator to configure code access security policy to grant the extended permissions to the code in the

specific assembly and not to the whole application.

For example, if your application needs to call unmanaged code, sandbox the unmanaged calls in a wrapper

assembly, so that an administrator can grant the **UnmanagedCodePermission** to the wrapper assembly and not

the whole application.

Note Sandboxing entails using a separate assembly and asserting security permissions to prevent full stack walks.

For more information about sandboxing unmanaged API calls, see "Unmanaged Code" in Chapter 8, "Code

Access Security in Practice."

Design Your Public Interface

Think carefully about which types and members form part of your assembly's public interface. Limit the

assembly's attack surface by minimizing the number of entry points and using a well designed, minimal public

interface.

Class Design Considerations

In addition to using a well defined and minimal public interface, you can further reduce your assembly's attack

surface by designing secure classes. Secure classes conform to solid object oriented design principles, prevent

inheritance where it is not required, limit who can call them, and which code can call them. The following

recommendations help you design secure classes:

- **Restrict class and member visibility**
- **Seal non base classes**
- **Restrict which users can call your code**
- **Expose fields using properties**

Restrict Class and Member Visibility

Use the **public** access modifier only for types and members that form part of the assembly's public interface.

This immediately reduces the attack surface because only public types are accessible by code outside the

assembly. All other types and members should be as restricted as possible. Use the **private** access modifier

wherever possible. Use **protected** only if the member should be accessible to derived classes and use **internal**

only if the member should be accessible to other classes in the same assembly.

Note C# also allows you to combine **protected** and **internal** to create a **protected internal** member to limit

access to the current assembly or derived types.

Seal Non-Base Classes

If a class is not designed as a base class, prevent inheritance using the **sealed** keyword as shown in the

following code sample.

```
public sealed class NobodyDerivesFromMe
{
}
```

For base classes, you can restrict which other code is allowed to derive from your class by using code access

security inheritance demands. For more information, see "Authorizing Code" in Chapter 8, "Code Access

Security in Practice."

Restrict Which Users Can Call Your Code

Annotate classes and methods with declarative principal permission demands to control which users can call

your classes and class members. In the following example, only members of the specified Windows group can

access the **Orders** class. A class level attribute like this applies to all class members.

Declarative principal

permission demands can also be used on individual methods. **Method level attributes override class level**

attributes.

```
[PrincipalPermission(SecurityAction.Demand,
```

```
Role=@"DomainName\WindowsGroup")]
```

```
public sealed class Orders()
```

```
{
```

```
}
```

QUESTION NO: 48

You are an application developer for Testkingprep.in. You are developing a Web service that will allow customers to access confidential information about their accounts. Client applications that use the Web service will provide an identifier and a password. The Web service will verify credentials by using a Microsoft SQL Server database. You must minimize the risk that account information can be intercepted during transmission.

You need to choose an authentication method.

What should you do?

A. Use Web Service Enhancements for Microsoft .NET (WSE) to implement the WS-Security specification.

- B. Require client applications to encrypt an identifier and password in the SOAP header of all requests.
- C. Use basic authentication over SSL/TLS.
- D. Use Forms authentication over SSL/TLS,

Answer: D

Explanation

Secure Sockets Layer / Transport Layer Security (SSL/TLS). This is most commonly used to secure the channel between a browser and Web server. However, it can also be **used to secure Web service messages and**

communications to and from a database server running Microsoft SQL Server 2000.

Know What to Secure

When a Web request flows across the physical deployment tiers of your application, it crosses a number of communication channels. A commonly used Web application deployment model is shown in Figure 1.



Figure 1. A typical Web deployment model

In this typical deployment model, a request passes through three distinct channels. The client-to-Web server

link may be over the Internet or corporate intranet and typically uses HTTP. The remaining two links are

between internal servers within your corporate domain. Nonetheless, all three links represent potential security

concerns. Many purely intranet-based applications convey security sensitive data between tiers; for example,

HR and payroll applications that deal with sensitive employee data.

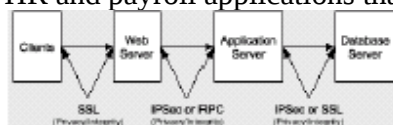


Figure 2 shows how each channel can be secured by using a combination of SSL, IPSec and RPC encryption.

Figure 2. A typical Web deployment model, with secure communications

The choice of technology depends on a number of factors including the transport protocol, end point technologies, and environmental considerations (such as hardware, operating system versions, firewalls, and so on).

SSL/TLS

SSL/TLS is used to establish an encrypted communication channel between client and server. The handshake

mechanism used to establish the secure channel is well documented and details can be found in the following

articles in the Microsoft Knowledge Base:

- Q257591, "Description of the Secure Sockets Layer (SSL) Handshake"
- Q257587, "Description of the Server Authentication Process During the SSL Handshake"
- Q257586, "Description of the Client Authentication Process During the SSL Handshake"

QUESTION NO: 49

You are an application developer for Testkingprep.in. You are developing a library assembly that communicates with a hardware device. Users will use the library assembly and the hardware device to develop custom applications. Users want their applications to be able to control access to the hardware device by using code access security policies. You need to ensure that users can code access security to restrict access to the hardware device.

What should you do?

- A. Create a custom permissions class.
Modify the library installer to install this permission and a new permission set.
- B. Create a custom evidence class.
- C. Create a custom security policy and a security policy development package.
Install this package at the same time as the library assembly.
- D. Create a publisher policy assembly for the library assembly.
Install this assembly with the library assembly.

Answer: A

Explanation

Implementing a Custom Permission

All permission objects must implement the `IPermission` interface. Inheriting from the `CodeAccessPermission`

class is the easiest way to create a custom permission because **`CodeAccessPermission`** implements

`IPermission` and provides most of the methods required for a permission. Additionally, you must implement the

`IUnrestrictedPermission` interface for all custom code access permissions. The custom permission class is

required for both imperative and declarative security support, so you should create it even if you plan to use

only declarative security.

Defining the Permission Class

To derive from the **`CodeAccessPermission`** class, you must override the following five key methods and

provide your own implementation:

- **`Copy`** creates a duplicate of the current permission object.

- **Intersect** returns the intersection of allowed permissions of the current class and a passed class.
- **IsSubsetOf** returns **true** if a passed permission includes everything allowed by the current permission.
- **FromXml** decodes an XML representation of your custom permission.
- **ToXml** encodes an XML representation of your custom permission.

The **IUnrestrictedPermission** interface requires you to override and implement a single method called

IsUnrestrictedPermission. In order to support the **IUnrestrictedPermission** interface, you must implement some system, such as a Boolean value that represents the state of restriction in the current object, to define whether the current instance of the permission is unrestricted.

Creating Your Own Code Access Permissions

The .NET Framework supplies a set of code access permission classes designed to help protect a specific set of resources and operations, focusing on those resources exposed by the .NET Framework. These permission

classes are described briefly in the Permissions topic and in detail in the reference documentation for each

permission class. For most environments, the built-in code access permissions are adequate. However, in some

situations, it might make sense to define your own code access permission class. This topic discusses when,

why, and how to define custom code access permission classes.

If you are defining a component or class library that accesses a resource that is not covered by the built-in

permission classes but needs to be protected from unauthorized code, you should consider creating a custom

code access permission class. If you want to be able to make declarative demands for your custom permission,

you must also define an attribute class for the permission. Providing these classes and making demands for the

permission from within your class library enables the runtime to prevent unauthorized code from accessing that

resource and enables an administrator to configure access rights.

There are other situations in which a custom permission might be appropriate. When a built-in code access

permission class protects a resource but does not sufficiently control access to that resource, you might need a

custom code access permission. For example, an application might use personnel records for which each

employee record is stored in a separate file; in such a case, read and write access could be controlled

independently for different types of employee data. An internal management tool could be authorized to read

certain sections of an employee's personnel file but not to modify those sections. In fact, it might not even be allowed to read some sections.

Custom code access permissions are also appropriate in cases where a built-in permission exists but is not defined in a way that enables it to protect the resource appropriately. For example, there might be a case in which there is UI functionality, such as the ability to create menus, that must be protected but is not protected by the built-in `UIPermission` class. In that case, you could create a custom permission to protect the ability to create menus.

Wherever possible, permissions should not overlap. Having more than one permission protecting a resource presents a significant problem for administrators, who must then be sure to deal appropriately with all the overlapping permissions every time they configure the rights to access that resource. Implementing a custom code access permission involves the following steps, some of which are optional. Each step is described in a separate topic.

1. Design the `Permission` class.
2. Implement the `IPermission` and `IUnrestrictedPermission` interfaces.
3. Implement the `ISerializable` interface, if necessary for performance or to support special data types.
4. Handle XML encoding and decoding.
5. Add support for declarative security, by implementing an **Attribute** class.
6. Demand custom permission for your permission, where appropriate.
7. Update security policy to be aware of the custom permission.

QUESTION NO: 50

You are an application developer for Testkingprep.in. You are implementing an ASP.NET Web application that uses Forms authentication. User names and passwords are stored in a Microsoft SQL Server database. The application includes the following method, which returns a value of true if the user provides a username and password that are found in the database.

```
private bool VerifyPassword(string userName, string password)
```

You configure your application to redirect unauthenticated requests to a page named `Logon.aspx`. This page includes text boxes for entering a user name and password, and it includes a `Logon` button.

You need to write the code to authenticate a user.

What should you do?

- A. Add the following code to the Click event handler for the **Logon** button.
- ```
if (VerifyPassword(txtUserName.Text,txtPassword.Text)) {
FormsAuthenticationTicket authTicket =
new FormsAuthenticationTicket(txtUserName.Text,false,30);
```

```

string encTicket = FormsAuthentication.Encrypt(authTicket);
HttpCookie authCookie =
new HttpCookie(FormsAuthentication.FormsCookieName,
encTicket);
Response.Cookies.Add(authCookie);
FormsAuthentication.RedirectFromLoginPage(txtUserName.Text, false);
}

```

B. Add the following code to the Click event handler for the Logon button.

```

if (VerifyPassword(txtUserName.Text,txtPassword.Text)) {
FormsAuthentication.Authenticate(txtUserName.Text,txtPassword.Text)
FormsAuthentication.RedirectFromLoginPage(txtUserName.Text, false);
}

```

C. Add the following code to the Application\_OnAuthenticate event handler.

```

string userName = (string) Context.Session["UserName"];
string password = (string) Context.Session["Password"];
if (VerifyPassword(userName,password)) {
FormsAuthenticationTicket authTicket =
new FormsAuthenticationTicket(userName,false,30);
string encTicket = FormsAuthentication.Encrypt(authTicket);
HttpCookie authCookie =
new HttpCookie(FormsAuthentication.FormsCookieName,
encTicket);
Response.Cookies.Add(authCookie);
FormsAuthentication.RedirectFromLoginPage(userName, false);
}

```

D. Add the following code to the Application\_OnAuthenticate event handler.

```

string userName = (string) Context.Session["UserName"];
string password = (string) Context.Session["Password"];

```

**Answer: B**

### Explanation

#### **FormsAuthentication.RedirectFromLoginPage Method (String, Boolean)**

Redirects an authenticated user back to the originally requested URL.

#### **Overloads Public Shared Sub RedirectFromLoginPage( \_**

**ByVal** *userName* As String, \_

**ByVal** *createPersistentCookie* As Boolean \_

#### **Parameters**

*userName*

Name of the user for cookie authentication purposes. This does not need to map to an account name and

will be used by URL Authorization.

*createPersistentCookie*

Specifies whether or not a durable cookie (one that is saved across browser sessions) should be issued.

#### **Remarks**

The **RedirectFromLoginPage** method redirects to the return URL key specified in the query string. For example, in the URL `http://www.contoso.com/login.aspx?ReturnUrl=caller.aspx`, `caller.aspx` is the return URL that **RedirectFromLoginPage** redirects to. If the return key does not exist,

**RedirectFromLoginPage** redirects to `Default.aspx`. ASP.NET automatically adds the return URL when

the browser is redirected to the login page specified in the **loginUrl** attribute in the `<forms>` Element

configuration directive. The method issues an authentication ticket and does a `SetForms` with the ticket, using

the appropriately configured cookie name for the application as part of the redirect response.

#### **FormsAuthenticationTicket Class**

Provides a means of creating and reading the values of a forms authentication cookie (containing an

authentication ticket) as used by `FormsAuthenticationModule`. This class cannot be inherited

#### **QUESTION NO: 51**

**You are an application developer for Testkingprep.in. You develop an ASP.NET Web application that is installed on a server named Testkingprep1. Testkingprep1 has IIS 5.0 installed. The Web application is configured to use Anonymous authentication in IIS. The Web.config file contains the following code segment.**

```
<authentication mode="Windows" />
```

```
<identity impersonate="true" />
```

**The Machine.config file contains the following code segment.**

```
<authentication mode="Windows" />
```

**The application implements security based on the following code segment.**

```
string myIdentity;
int validAccess = 0;
if (myIdentity == @"Testkingprep1\ASPNET") {
 validAccess = 1;
}
```

**For testing purposes, you display the value of the validAccess variable in a label.**

**When you run the application, you discover that the value of validAccess is 0.**

**You need to ensure that validAccess has a value of 1.**

**What should you do?**

A. Replace the code segment in the `Web.config` file with the following code segment.

```
<authentication mode="Windows" />
```

```
<identity impersonate="false" />
```

B. Replace the code segment in the `Web.config` file with the following code segment.

```
<authentication mode="Forms" />
```

```
<identity impersonate="true" />
```

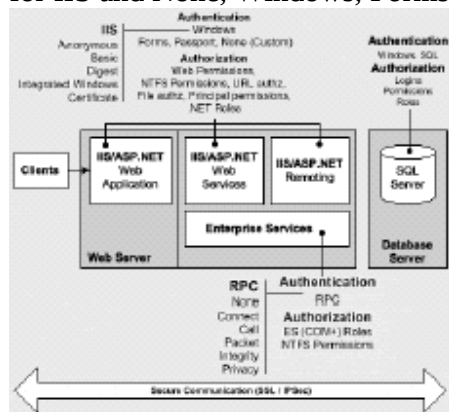
- C. Ask a network administrator to change the authentication mode of the Web application to Integrated Windows authentication.
- D. Ask a network administrator to change the authentication mode of the Web application to basic authentication.

**Answer: A**

### Explanation

Microsoft's Security Architecture provides for 5 Authentication modes in IIS and 4 Authentication modes in ASP.NET

These include Anonymous, Basic, Digest, Windows Integrated (Kerberos/NTLM) and Certificate authentications for IIS and None, Windows, Forms and Passport for ASP.NET



For each of the authentication modes in ASP.NET, 'Impersonate' can be set to False or True. Setting Impersonate to False results in ASP.NET applications using the default system account ASPNET to access resources. Setting Impersonate to True results in ASP.NET using the identity passed by IIS(IUSR\_machinename , domain\username , etc).

\* HttpContext = This is the authenticated Web client.

\* WindowsIdentity = This is the identity of the security context of the currently executing Win32 thread.

\* Thread = This is the identity of the currently executing .NET thread which rides on top of the Win32 thread

Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication

Principal objects implement the **IPrincipal** interface and represent the security context of the user on whose

behalf the code is running. The principal object includes the user's identity (as a contained **Identity** object) and

any roles to which the user belongs.

ASP.NET provides the following principal and identity object implementations:

- **WindowsPrincipal** and **WindowsIdentity** objects represent users who have been authenticated with Windows authentication. With these objects, the role list is automatically obtained from the set of Windows groups to which the Windows user belongs.
  - **GenericPrincipal** and **GenericIdentity** objects represent users who have been authenticated using Forms authentication or other custom authentication mechanisms. With these objects, the role list is obtained in a custom manner, typically from a database.
  - **FormsIdentity** and **PassportIdentity** objects represent users who have been authenticated with Forms and Passport authentication respectively.
- The following tables illustrate, for a range of IIS authentication settings, the resultant identity that is obtained from each of the variables that maintain an **IPrincipal** and/or **Identity** object. The following abbreviations are used in the table:
- **HttpContext** = **HttpContext.Current.User**, which returns an **IPrincipal** object that contains security information for the current Web request. This is the authenticated Web client.
  - **WindowsIdentity** = **WindowsIdentity.GetCurrent()**, which returns the identity of the security context of the currently executing Win32 thread.
  - **Thread** = **Thread.CurrentPrincipal** which returns the principal of the currently executing .NET thread which rides on top of the Win32 thread.

**Table 1. IIS anonymous authentication**

**Web.config Settings Variable Location Resultant Identity**

```

<identity impersonate="true"/>
<authentication mode="Windows" />
HttpContext
WindowsIdentity
Thread
-
MACHINE\IUSR_MACHINE
-
<identity impersonate="false"/>
<authentication mode="Windows" />
HttpContext
WindowsIdentity
Thread
-
MACHINE\ASPNET
-
<identity impersonate="true"/>

```

<authentication mode="Forms" />

HttpContext

WindowsIdentity

Thread

Name provided by user

MACHINE\IUSR\_MACHINE

Name provided by user

<identity impersonate="false"/>

<authentication mode="Forms" />

HttpContext

WindowsIdentity

Thread

Name provided by user

MACHINE\ASPNET

Name provided by user

## **Table 2. IIS basic authentication**

### **Web.config Settings Variable Location Resultant Identity**

<identity impersonate="true"/>

<authentication mode="Windows" />

HttpContext

WindowsIdentity

Thread

Domain\UserName

Domain\UserName

Domain\UserName

<identity impersonate="false"/>

<authentication mode="Windows" />

HttpContext

WindowsIdentity

Thread

Domain\UserName

MACHINE\ASPNET

Domain\UserName

<identity impersonate="true"/>

<authentication mode="Forms" />

HttpContext

WindowsIdentity

Thread

Name provided by user

Domain\UserName

Name provided by user

<identity impersonate="false"/>

<authentication mode="Forms" />

HttpContext

WindowsIdentity

Thread

Name provided by user

MACHINE\ASPNET

Name provided by user

### **Table 3. IIS digest authentication**

#### **Web.config Settings Variable Location Resultant Identity**

<identity impersonate="true"/>

<authentication mode="Windows" />

HttpContext

WindowsIdentity

Thread

Domain\UserName

Domain\UserName

Domain\UserName

<identity impersonate="false"/>

<authentication mode="Windows" />

HttpContext

WindowsIdentity

Thread

Domain\UserName

MACHINE\ASPNET

Domain\UserName

<identity impersonate="true"/>

<authentication mode="Forms" />

HttpContext

WindowsIdentity

Thread

Name provided by user

Domain\UserName

Name provided by user

<identity impersonate="false"/>

<authentication mode="Forms" />

HttpContext

WindowsIdentity

Thread

Name provided by user

MACHINE\ASPNET

Name provided by user

### **Table 4: IIS integrated Windows**

#### **Web.config Settings Variable Location Resultant Identity**

<identity impersonate="true"/>

<authentication mode="Windows" />

HttpContext

WindowsIdentity

Thread

Domain\UserName

Domain\UserName

Domain\UserName  
<identity impersonate="false"/>  
<authentication mode="Windows" />

HttpContext

WindowsIdentity

Domain\UserName

MACHINE\ASPNET

hread Domain\UserName

<identity impersonate="true"/>

<authentication mode="Forms" />

HttpContext

WindowsIdentity

Thread

Name provided by user

Domain\UserName

Name provided by user

<identity impersonate="false"/>

<authentication mode="Forms" />

HttpContext.

WindowsIdentity

Thread

Name provided by user

MACHINE\ASPNET

Name provided by user

The machine configuration file, Machine.config, contains settings that apply to an entire computer. This file is

located in the %runtime install path%\Config directory. Machine.config contains configuration settings for

machine-wide assembly binding, built-in remoting channels, and ASP.NET.

The configuration system first looks in the machine configuration file for the

<appSettings> element and other

configuration sections that a developer might define. It then looks in the application configuration file. To keep

the machine configuration file manageable, it is best to put these settings in the application configuration file.

However, putting the settings in the machine configuration file can make your system more maintainable. For

example, if you have a third-party component that both your client and server application uses, it is easier to put

the settings for that component in one place. In this case, the machine configuration file is the appropriate place

for the settings, so you don't have the same settings in two different files.

ASP.NET configuration, of which security is a part, has a hierarchical architecture. All configuration

information for ASP.NET is contained in files named Web.config and Machine.config.

Web.config can be

placed in the same directories as the application files. The Machine.config file is in the Config directory of the install root. Subdirectories inherit a directory's settings unless overridden by a Web.config file in the subdirectory. In a Web.config file, there are sections for each major category of ASP.NET functionality.

#### **QUESTION NO: 52**

**You are an application developer for Testkingprep.in. You develop an assembly for a Windows-based application that is installed on all computers in Testkingprep. A written company policy states that access to the assembly is not allowed from computers at one office named Location1. Access to the assembly is allowed from all other locations in the company. One employee at Location1 is granted an exception to this policy and requires access to the assembly. You need to enable the one employee to access the assembly. You plan to achieve this goal by using the Microsoft .NET Framework Configuration tool. What should you do?**

- A. Grant the FullTrust permission for the assembly at the enterprise level.  
Deny all permissions for the assembly at the user level.
- B. Grant the FullTrust permission for the assembly at the enterprise level and the user level.  
Deny all permissions for the assembly at the machine level.
- C. Grant the FullTrust permission for the assembly at the machine level.  
Deny all permissions for the assembly at the user level and the enterprise level.
- D. Grant the FullTrust permission for the assembly at the machine level and the user level.  
Deny all permissions for the assembly at the enterprise level.

**Answer: B**

#### **Explanation**

##### **User Policy Administration**

User policy is the lowest administrable policy level. Every user has an individual user policy configuration file. Any changes made to this policy level are applicable only to the current logged-on user. The user policy level is restricted in what it can specify. Because this level is configurable by the current logged-on user, enterprise level policy administrators should be aware that the user might potentially alter any policy changes made on the user policy level. The user policy level is not able to give more permissions to an assembly than is specified in the higher policy levels. However, the user policy level is allowed to decrease permissions, which might potentially cause applications to stop functioning properly. If the **LevelFinal** attribute is applied to a code group on the machine or enterprise level,

the user level is not allowed to tighten policy decisions that have been made on those levels. User level administration is appropriate in some situations to tightening security. For example, a user might decide to tighten security policy for assemblies that originate from the local intranet zone if untrusted code is found there. You might consider administering policy on this level when you are a user on a corporate network and believe that the security settings are not tight enough.

### **Machine Policy Administration**

The machine policy level holds most of the default security policy. All machine and domain administrators have access to the machine configuration files. Machine administrators can set policy that excludes modification from the user level but not from the enterprise level.

You might consider administering security policy on this level in the following situations:

- You are not on a network or are on a network without a domain controller.
- The computer you are administering serves a unique function. For example, if you are administering a public computer that is used for general Internet access by several people in a semi-public setting, you might want to have a unique machine policy, because the computer serves a unique function. Additionally, you might want to produce a specific machine policy that considers the security needs of specialized computers, like the servers in your enterprise.

### **Enterprise Policy Administration**

The enterprise policy level affects every computer and user on the network and can only be administered by enterprise or domain administrators. See the section on Deploying Security Policy for information on deployment strategies.

Because the runtime evaluates enterprise policy first, you can apply the `LevelFinal` attribute to a code group on this level to exclude the lower levels from making policy changes. If you do not apply the `LevelFinal` attribute to code groups on this level, administrators of lower security levels will be able to assign more permissions to applications without your knowledge and potentially create security vulnerabilities. You might consider administering policy on this level when every person in your enterprise uses an application and you want to make sure that it always receives sufficient permission to run.

### **QUESTION NO: 53**

**You are an application developer for Testkingprep.in. A developer in Testkingprep develops a component named TestkingprepComponent. You deploy TestkingprepComponent to a server. After you execute TestkingprepComponent,**

**you receive an error message that reads in part:**

**“System.Security.Policy.PolicyException.”**

**You need to find out the cause of the error.**

**What should you do?**

- A. View the NTFS permissions of MyComponent.dll to find out the required permissions.
- B. View the permissions requested by the component at run time by using the Microsoft CLR Debugger.
- C. View the permissions requested by the component at run time by using the Runtime Debugger.
- D. View the required permissions for the component by using the Permissions View tool.

**Answer: D**

### **Explanation**

#### **PolicyException Class**

The exception that is thrown when policy forbids code to run.

For a list of all members of this type, see PolicyException Members.

System.Object

System.Exception

System.SystemException

**System.Security.Policy.PolicyException**

<Serializable>

**Public Class PolicyException**

**Inherits SystemException**

#### **Remarks**

This exception is typically thrown when the code requests more permissions than the policy will grant or the policy is configured to prohibit running the code

#### **Permissions View Tool (Permview.exe)**

The Permissions View tool is used to view the minimal, optional, and refused permission sets requested by an

assembly. Optionally, you can use Permview.exe to view all declarative security used by an assembly.

**permview** [/output *filename*] [/decl] *manifestfile*

#### **Argument Description**

*manifestfile* The file that contains the assembly's manifest. The manifest can be either a standalone file or it can be incorporated in a portable executable (PE) file. The extension for this file will usually be .exe or .dll, but it could also be .scr, or .ocx.

#### **Option Description**

**/decl** Displays all declarative security at the assembly, class, and method level for the assembly specified by *manifestfile*. This includes permission requests as well as demands, asserts, and all other security actions that can be applied declaratively. It does not refer to other assemblies linked to the specified assembly.

**/h[elp]** Displays command syntax and options for the tool.

**/output filename** Writes the output to the specified file. The default is to display the output to the console.

**/?** Displays command syntax and options for the tool.

### Remarks

Developers can use Permview.exe to verify that they have applied permission requests correctly to their code.

Additionally, users can run Permview.exe to determine the permissions an assembly requires to execute. For

example, if you run a managed executable and get the error,

"System.Security.Policy.PolicyException: Failed to

acquire required permissions," you can use Permview.exe to determine the permissions the code in your

executable must receive before it will execute.

### Examples

The following command displays the permissions requested by the assembly myAssembly.exe to the console.

```
permview myAssembly.exe
```

If myAssembly.exe contains a minimum request for **FullTrust**, the following output appears.

Microsoft (R) .NET Framework Permission Request Viewer. Version 1.0.2204.18

Copyright (C) Microsoft

Corp. 1998-2000

minimal permission set:

```
<PermissionSet class="System.Security.PermissionSet" version ="1">
```

```
<Unrestricted/>
```

```
</PermissionSet>
```

optional permission set:

Not specified

refused permission set:

Not specified

he following command displays all declarative security on the assembly myAssembly.exe to the console.

This command displays the method level security demand.

```
permview /decl myAssembly.exe
```

The following output appears.

Microsoft (R) .NET Framework Permission Request Viewer. Version 1.0.2204.18

Copyright (C) Microsoft

Corp. 1998-2000

Assembly RequestMinimum permission set:

```
<PermissionSet class="System.Security.PermissionSet" version ="1">
```

```
<Unrestricted/>
```

```
</PermissionSet>
```

Method A::myMethod() LinktimeCheck permission set:

```
<PermissionSet class="System.Security.PermissionSet" version="1">
<Permission class="System.Security.Permissions.ReflectionPermission, mscorlib,
Ver=1.0.2204.2, Loc=",
SN=03689116d3a4ae33" version="1">
<MemberAccess/>
</Permission>
</PermissionSet>
```

The following command writes the permissions requested by the assembly myAssembly.exe to the file myOutputFile. permview /output myOutputFile myAssembly.exe

#### **QUESTION NO: 54**

**You are an application developer for Testkingprep.in. You are developing the business layer of a three-tier**

**Web application. The application uses Forms authentication. File access permissions are assigned based on the role of the user. The authenticated user names and roles are stored in a Microsoft SQL Server database. When a user is authenticated by the server, the user's cryptographically random session token and role are stored in a cookie on the client computer and used for access to other pages.**

**You want users who are members of a role named Editor to have Read permission and Write permission for the application files. You want users who are members of a role named Reader to have only Read permission for the files. You create a method named OpenFile to pass the name and role of the current user along with the name of a file. This information is used to open and return a file. The method is contained in the following code segment.**

```
public Stream OpenFile(HttpCookieCollection userInfo, string file) {
switch (userInfo["Role"].Value) {
case "Editor":
return new FileStream(file, FileMode.Open, FileAccess.Read);
}}
```

**During a security review, you discover that some users will receive Write permission when they shouldn't.**

**You need to prevent unauthorized users from modifying files.**

**What should you do?**

- A. Change the application to use Windows authentication.
- B. Create a restrictive discretionary access control list (DACL) entry for each file.
- C. Verify the role by using the word Reader, instead of relying on the default case.
- D. Retrieve the user's role information from the database instead of from the cookie.

**Answer: D**

#### **Explanation**

Cookies are a prime target for session high-jacking. There should be limited information stored in them as they

can be manipulated client-side if stored there. Since the role is stored here in plain text it can just be changed to a higher role. The question only ask for one option but it specifically states that one role has read/write and the other only has read, but the code only checks for the read/write role, which would mean everyone using the code would at least have read when in only the one role has read only. It would be best to not only pull the role from a more secure/securable source other than the cookie but also check for the roles in the case select.

### **Bulletproof persistent cookies to increase security**

Web browser cookies can enhance the user experience by providing additional functionality and ease of use.

However, from an administration point of view, cookies are a security concern. Encrypt your cookies with this simple technique.

Cookies offer an excellent way to keep small bits of information about the user readily available so that they don't have to be looked up again. They can allow you to keep users' numeric IDs handy instead of their logon names, which makes getting back to their security and authorization easier and quicker. However, cookies represent a dangerous risk: Users may choose to tamper with the information and see what havoc it might cause.

### **Risks of cookie tampering**

To understand how using cookies can create huge security risks, consider a site that stores the user's ID in a database in a cookie. The cookie is persistent, and the site never validates the information in the cookie. It's assumed to be correct. The user goes to the site and logs in. He or she looks at the cookie file and determines that only an ID is being stored in it. The user resets the number in the file to 1 and logs back into the site.

For most sites, the super user is the first ID inserted into the database, and it's usually never disabled. If the site doesn't validate the value from the cookie, the user has become a complete administrator with a trivial amount of text editing. Any unchecked information placed in a cookie can represent a potential security problem.

### **QUESTION NO: 55**

**You are an application developer for Testkingprep.in. You are modifying an application that was developed by another developer. The application was developed by using the Microsoft .NET Framework. The application accepts user input into a variable named `userinput` and saves information in a Microsoft SQL Server database. The application uses the following code to construct a SQL query**

`string sqlquery`

`sqlquery = "SELECT FROM Table1 WHERE ColumnA = " +`

```
userinput = "";
```

**You need to improve the security of the application to reduce the likelihood of SQL injection attacks from user input.**

**Which three actions should you perform for all user input? (Each correct answer presents part of the solution. Choose three)**

- A. Remove double hyphens (--) from the input.
- B. Encode the input by using the HttpUtility.HtmlEncode method.
- C. Decode the input by using the HttpUtility.HtmlDecode method.
- D. Limit the length to the exact size of the SQL Server column where the input will be stored.
- E. Replace single quotation marks (') with two single quotation marks ("").
- F. Convert all characters to Unicode.

**Answer: A, D, E**

### **Explanation**

#### **Preventing SQL Injection Attacks**

If you design your scripts and applications with care, SQL injection attacks can be avoided most of the time.

There are a number of things that we as developers can do to reduce our site's susceptibility to attack. Here's a list (in no particular order) of our options:

#### ***Limit User Access***

The default system account (sa) for SQL server 2000 should never be used because of its unrestricted nature.

You should always setup specific accounts for specific purposes.

For example, if you run a database that lets users of your site view and order products, then you should set up a user called webUser\_public that has SELECT rights on the products table, and INSERT rights only on the orders table.

If you don't make use of extended stored procedures, or have unused triggers, stored procedures, user-defined functions, etc, then remove them, or move them to an isolated server. Most extremely damaging SQL injection attacks attempt to make use of several extended stored procedures such as xp\_cmdshell and xp\_grantlogin, so by removing them, you're theoretically blocking the attack before it can occur.

#### ***Escape Quotes***

As we've seen from the examples discussed above, the majority of injection attacks require the user of single quotes to terminate an expression. By using a simple replace function and converting all single quotes to two single quotes, you're greatly reducing the chance of an injection attack succeeding.

Using ASP, it's a simple matter of creating a generic replace function that will handle the single quotes automatically, like this:

```
<%
function stripQuotes(strWords)
stripQuotes = replace(strWords, "'", "'")
end function
%>
```

Now if we use the stripQuotes function in conjunction with our first query for example, then it would go

from this:

```
select count(*) from users where userName='john' and
userPass="" or 1=1 --'
```

...to this:

```
select count(*) from users where userName='john' and
userPass="" or 1=1 --'
```

This, in effect, stops the injection attack from taking place, because the clause for the WHERE query now

requires both the userName and userPass fields to be valid.

### ***Remove Culprit Characters/Character Sequences***

As we've seen in this article, certain characters and character sequences such as ;, --, select, insert and

xp\_ can be used to perform an SQL injection attack. By removing these characters and character sequences

from user input before we build a query, we can help reduce the chance of an injection attack even further.

As with the single quote solution, we just need a basic function to handle all of this for us:

```
<%
function killChars(strWords)
dim badChars
dim newChars
badChars = array("select", "drop", ";", "--", "insert",
"delete", "xp_")
newChars = strWords
for i = 0 to uBound(badChars)
newChars = replace(newChars, badChars(i), "")
next
killChars = newChars
end function
%>
```

Using stripQuotes in combination with killChars greatly removes the chance of any SQL injection

attack from succeeding. So if we had the query:

```
select prodName from products where id=1; xp_cmdshell 'format
c: /q /yes '; drop database myDB; --
```

and ran it through stripQuotes and then killChars, it would end up looking like this:

```
prodName from products where id=1 cmdshell "format c:
```

```
/q /yes " database myDB
```

...which is basically useless, and will return no records from the query.

### ***Limit the Length of User Input***

It's no good having a text box on a form that can accept 50 characters if the field you'll compare it against can

only accept 10. By keeping all text boxes and form fields as short as possible, you're taking away the number of

characters that can be used to formulate an SQL injection attack.

If you're accepting a querystring value for a product ID or the like, always use a function to check if the value is

actually numeric, such as the IsNumeric() function for ASP. If the value isn't numeric, then either raise an

error or redirect the user to another page where they can choose a product.

Also, always try to post your forms with the method attribute set to POST, so clued-up users don't get any ideas

--- they might if they saw your form variables tacked onto the end of the URL.

## **SQL Server 2000 SP3 Security Features and Best Practices: Security Best Practices Checklist**

Administrator Checklist

### **Setting Up the Environment Prior to Installation**

Physical security • Ensure the physical security of your server.

Firewalls • Put a firewall between your server and the Internet.

- Always block TCP port 1433 and UDP port 1434

on your perimeter firewall. If named instances are listening on additional ports, block those too.

- In a multi-tier environment, use multiple firewalls to create screened subnets.

Isolation of services • Isolate services to reduce the risk that a compromised service could be used to compromise

### **Setting Up the Environment Prior to Installation**

others.

- Never install SQL Server on a domain controller.

- Run separate SQL Server services under separate Windows accounts.

- In a multi-tier environment, run Web logic and business logic on separate computers.

Service accounts • Create Windows accounts with the lowest possible privileges for running SQL Server services.

File System • Use NTFS.

- Use RAID for critical data files.

### **Installation**

Latest version and service pack • Always install the latest service packs and security patches.

Service accounts • Run SQL Server services with the lowest possible

privileges.

- Use Enterprise Manager to associate services with Windows accounts.

Authentication mode • Require Windows Authentication for connections to SQL Server.

Strong passwords • Always assign a strong password to the **sa** account, even when using Windows Authentication.

- Always use strong passwords for all SQL Server accounts.

### **Configuration Options and Settings After Installation**

Delete or secure old setup files • Delete or archive the following files after installation: sqlstp.log, sqlsp.log, and setup.iss in

the <systemdrive>:\Program Files\Microsoft SQL Server\MSSQL\Install folder for a default

installation, and the <systemdrive>:\Program Files\Microsoft SQL Server\MSSQL\$<Instance Name>\Install folder for named instances.

- If the current system is an upgrade from SQL Server 7.0, delete the following files: setup.iss in the %Windir% folder, and sqlsp.log in the Windows Temp folder.

Choose static ports for named instances • Assign static ports to named instances of SQL Server.

Set login auditing level • Set login auditing level to **failure** or **all**.

Enable security auditing • Enable security auditing of **Sysadmin** actions, fixed role membership changes, all login related activity, and password changes.

- After selecting appropriate auditing options, you should script the audit, wrap it in a stored procedure, and mark that stored procedure for AutoStart.

Secure **sa** even in Windows Authentication Mode • Assign a strong password to the **sa** account, even

on servers that are configured to require Windows Authentication.

### **Configuration Options and Settings After Installation**

Remove sample databases • Remove sample databases from production servers.

### **Secure Operation**

Security model • Learn to work with the SQL Server security model.

Backup policy • Back up all data regularly and store copies in a secure off-site location.

- Test your disaster recovery system.

Surface and feature reduction • Reduce the surface area of your system that is exposed to attack by running only those services

and features needed in your environment.

Administrator reduction • Restrict membership of the **sysadmin** fixed server role to a few trusted individuals.

Strong passwords • Ensure that you use complex passwords for all SQL Server accounts.

Cross database ownership chaining • Disable cross database ownership chaining if your system does not use it.

Xp\_cmdshell • By default, only members of the sysadmin role can execute xp\_cmdshell. You should not change this default.

- Do not grant execute permission on xp\_cmdshell to users who are not members of the sysadmin role.

Encryption • Install a certificate to enable SSL connections.

- Certificates should use the fully-qualified DNS

### **Secure Operation**

name of the server.

- Use the SQL Server service account to encrypt database files with EFS.

- If your application requires data encryption, consider using the products of such vendors as Protegrity and Application Security Inc.

Roles and groups • Collect users into SQL Server roles or Windows groups to simplify permissions administration.

Permissions • Never grant permissions to the **public** database role.

Distributed queries • When setting up SQL Server in an environment that supports distributed queries, use linked servers rather than remote servers.

- Allow linked server access only to those logins that need it.

- Disable ad hoc data access on all providers except SQL OLE DB, for all users except members of the **sysadmin** fixed server role.

- Allow ad hoc data access only on trusted providers.

Guest accounts • Do not enable the guest account.

Service accounts • If you need to change the account associated with a SQL Server service, use SQL Server Enterprise Manager.

- If you change multiple services, you must apply

### **Secure Operation**

the changes to each service separately using Enterprise Manager.

### **Recommended Periodic Administrative Procedures**

Microsoft Baseline Security Analyzer • Add MBSA to your weekly maintenance schedule,

and follow up on any security recommendations that it makes.

Scanning logins • Periodically scan for accounts with NULL passwords and remove them or assign them strong passwords.

- Delete unused accounts.

Enumerate fixed role membership • Periodically scan fixed server and database roles to ensure that membership is only granted to trusted individuals.

Start-up procedures • Verify the safety of stored procedures that have been marked for AutoStart.

Login-to-user mapping • Ensure that the mapping between database users and logins at the server level is correct.

- Run **sp\_change\_users\_login** with the **report** option regularly to ensure that the mapping is as expected.

Direct catalog updates • Do not allow direct catalog updates.

Cross database ownership chaining • Use **sp\_dboption** to enumerate and validate databases for which cross database ownership chaining has been enabled.

### **Best Practices for Patching Instances**

Instance detection and enumeration • Keep an inventory of all versions, editions, and languages of SQL Server for which you are responsible.

- Include instances of MSDE in your inventory.

- Use SQL Scan and SQL Check, available from the Microsoft Web site, to scan for instances of SQL Server within your domain.

Bulletins • Subscribe to Microsoft security bulletins.

Patch application • Maintain test systems that match the configuration of your production systems, and are readily available for testing new patches.

- Test patches carefully before applying them to production systems.

- Consider patching development systems with relatively little testing.

### **Developer Checklist**

In addition to all of the items above, the following should be considered best practices for developers.

### **General**

Use ownership chaining effectively • Use ownership chaining within a single database to simplify permissions management.

- Avoid using cross database ownership chaining when possible.

- If you must use cross database ownership chaining, ensure that the two databases are always deployed

as a single administrative unit.

### **General**

Use roles to simplify permission management and ownership

- Assign permissions to roles rather than directly to users.

- Objects may be owned by roles, rather than directly by users, if you want to avoid application changes when the owning user is dropped.

Turn on encryption (SSL or IPSEC) • Enable encrypted connections to your server, and consider allowing only encrypted connections.

- When allowing SQL Server Authentication, you are strongly urged to encrypt either the network layer with IPsec or the session with SSL.

Do not propagate SQL Server errors back to user • Your application should not return SQL Server

errors to the end user. Log them instead, or transmit them to the system administrator.

Prevent SQL injection • Defend against SQL injection by validating all user input before transmitting it to the server.

- Limit the scope of possible damage by permitting only minimally privileged accounts to send user input to the server.

- Run SQL Server itself with the least necessary privileges.

### **Multi-tier Options**

Same/trusted domain (complete Windows Authentication)

If the application server and the database server are within the same domain, or within trusted domains, you should use Windows Authentication and configure for "full provisioning" in which all client contexts are tunneled to SQL Server. This makes it possible to audit all users who access SQL Server, enables Windows

### **Multi-tier Options**

security policy enforcement, and makes it unnecessary to store credentials in the middle tier. In this scenario, the client connects to the application server, which in turn impersonates the client and connects to SQL Server.

- Every user on the application server must have a valid Windows login on the database server and delegation must be enabled.

- All systems interacting in this scenario, including

the Domain Controller, must run Windows 2000 or higher.

- The account the application is running under must be trusted for delegation (that is, the Active Directory option **Account is trusted for delegation** must be turned on for this account).
- The client account must be able to be delegated (ensure that the Active Directory user account option **Account is trusted and cannot be delegated** is unchecked).
- The application service must have a valid Service Principal Name (SPN).

**Note:** Full provisioning is not recommended in cross-enterprise or Internet-scale installations, when your security plan calls for minimizing user access to the database server, or in enterprises with policies prohibiting delegation.

Mixed scenario (partial Windows Authentication) When the Internet-facing tier does not have an

individual Windows domain account for every possible user, the recommended scenario is to divide authentication into stages. The outer tier (which authenticates users) should use SSL to encrypt at least

### **Multi-tier Options**

credentials, if not the entire session. It should connect to the database server using Windows Authentication, forwarding transaction information under a separate security context that is low privileged, with only the permissions necessary to perform its function. This effectively uses the middle tier as an additional layer of defense between your server and the Internet.

**Note:** Using SQL Server Authentication between the middle tier and SQL Server is not recommended, because of the need to store credentials. If you must use SQL Server Authentication between the middle tier and SQL Server, you should create several accounts, with different levels of privileges corresponding to different classes of users. This requires that you add logic to the middle tier to allocate connections according to the desired privilege level. Different non-trusted domains or no domains (no Windows Authentication)

When Windows Authentication between tiers is not possible, you should require SSL encryption of the login sequence. Encrypting the entire session is

preferable.

- You should also use DPAPI to encrypt credentials that must be stored.
- You should store encrypted credentials in a registry key protected with an ACL.

#### Software Vendor Checklist

In addition to all of the items above, the following security development practices have proven useful in

increasing the quality and security of code in various development environments.

#### Security Processes

Understanding various security issues • Ensure that members of your development team understand major security issues: current threats, security trends, changing security environments, and attack scenarios.

#### Security Processes

- Require relevant security training for all developers and testers.
- Increase the awareness of issues like cross-site scripting, buffer overflows, SQL injection, and dangerous APIs.
- Identify specific categories of threats that apply to your product — for example, denial of service, escalation of privileges, spoofing, data tampering, information disclosure and repudiation.
- Analyze security threats to your product, component-by-component.
- Create a security threat checklist based on your product.
- Add security reviews to all stages (from design to testing) of your product development cycle.

MSDE installations If you distribute MSDE with your application, the following additional guidance applies:

- Install MSDE using "Windows security mode" as the default.
- Never install a blank **sa** password.
- When distributing MSDE to your customers, you should use the Microsoft-supplied installer rather than merge modules.
- When installing an instance of MSDE that will operate only as a local data store, you should disable the Server Net-Libraries.

#### Security Processes

- If your product includes MSDE, you should make this known to your customers. In the future, they

may need to install or accept MSDE-specific software updates.

- MSDE installs SQL Server Agent by default, but leaves the Service startup type to "Manual." If your application does not use SQL Server Agent, you should change this to "Disabled." Include security best practice information in your product documentation.

#### QUESTION NO: 56

**You are an application developer for Testkingprep.in. You develop an ASP.NET Web application that uses a database to keep track of hours worked by each employee. The application stores the account name of the interactive user in a variable named `userName`. The application uses the value of `userName` and data entered by each user to record the user name and hours worked. The application is configured to use Integrated Windows authentication in IIS. The `Web.config` file has Windows authentication configured and impersonation enabled. During a security review, you find out that the application is running under a user context that has more permissions than necessary. You need to increase the security of the application while maintaining current functionality.**

**What should you do?**

A. Ask a network administrator to enable basic authentication for the application in IIS and prompt the user

to enter the user's user name and password.

B. Ask a network administrator to enable digest authentication for the application in IIS and prompt the

user to enter the user's user name and password.

C. Change the `Web.config` file to set impersonation to **False**.

Add the following code to populate the `userName` variable with the user name of the interactive user.

```
string userName;
```

```
userName = HttpContext.Current.User.Identity.Name.ToString();
```

D. Change the `Web.config` file to set impersonation to **false**.

Add the following code to populate the `userName` variable with the user name of the interactive user.

```
WindowsIdentity myIdentity;
```

**Answer: C**

#### Explanation

Principal objects implement the **IPrincipal** interface and represent the security context of the user on whose

behalf the code is running. The principal object includes the user's identity (as a contained **Identity** object) and

any roles to which the user belongs.

ASP.NET provides the following principal and identity object implementations:

- **WindowsPrincipal** and **WindowsIdentity** objects represent users who have been authenticated with

Windows authentication. With these objects, the role list is automatically obtained from the set of Windows groups to which the Windows user belongs.

- **GenericPrincipal** and **GenericIdentity** objects represent users who have been authenticated using

Forms authentication or other custom authentication mechanisms. With these objects, the role list is obtained

in a custom manner, typically from a database.

- **FormsIdentity** and **PassportIdentity** objects represent users who have been authenticated with Forms

and Passport authentication respectively.

The following tables illustrate, for a range of IIS authentication settings, the resultant identity that is obtained

from each of the variables that maintain an **IPrincipal** and/or **Identity** object. The following abbreviations are used in the table:

- **HttpContext** = **HttpContext.Current.User**, which returns an **IPrincipal** object that contains security

information for the current Web request. This is the authenticated Web client.

- **WindowsIdentity** = **WindowsIdentity.GetCurrent()**, which returns the identity of the security context

of the currently executing Win32 thread.

- **Thread** = **Thread.CurrentPrincipal** which returns the principal of the currently executing .NET thread

which rides on top of the Win32 thread.

**Table 1. IIS anonymous authentication**

Web.config Settings	Variable	Location	Resultant Identity
---------------------	----------	----------	--------------------

<identity impersonate="true"/> <authentication mode="Windows" />	HttpContext		
	WindowsIdentity		
	Thread		
	-		

<identity impersonate="false"/> <authentication mode="Windows" />	HttpContext		
	WindowsIdentity		
	Thread		
	-		

-

```
<identity impersonate="true"/>
<authentication mode="Forms" />
```

HttpContext

WindowsIdentity

Thread

Name provided by user

MACHINE\IUSR\_MACHINE

Name provided by user

```
<identity impersonate="false"/>
```

```
<authentication mode="Forms" />
```

HttpContext

WindowsIdentity

Thread

Name provided by user

MACHINE\ASPNET

Name provided by user

## **Table 2. IIS basic authentication**

### **Web.config Settings Variable Location Resultant Identity**

```
<identity impersonate="true"/>
```

```
<authentication mode="Windows" />
```

HttpContext

WindowsIdentity

Thread

Domain\UserName

Domain\UserName

Domain\UserName

```
<identity impersonate="false"/>
```

```
<authentication mode="Windows" />
```

HttpContext

WindowsIdentity

Thread

Domain\UserName

MACHINE\ASPNET

Domain\UserName

```
<identity impersonate="true"/>
```

```
<authentication mode="Forms" />
```

HttpContext

WindowsIdentity

Thread

Name provided by user

Domain\UserName

Name provided by user

```
<identity impersonate="false"/>
```

```
<authentication mode="Forms" />
```

HttpContext

WindowsIdentity  
Thread  
Name provided by user  
MACHINE\ASPNET  
Name provided by user

**Table 3. IIS digest authentication**

**Web.config Settings Variable Location Resultant Identity**

<identity impersonate="true"/>  
<authentication mode="Windows" />  
HttpContext  
WindowsIdentity  
Thread  
Domain\UserName  
Domain\UserName  
Domain\UserName  
<identity impersonate="false"/>  
<authentication mode="Windows" />  
HttpContext  
WindowsIdentity  
Domain\UserName  
MACHINE\ASPNET  
Thread Domain\UserName  
<identity impersonate="true"/>  
<authentication mode="Forms" />  
HttpContext  
WindowsIdentity  
Thread  
Name provided by user  
Domain\UserName  
Name provided by user  
<identity impersonate="false"/>  
<authentication mode="Forms" />  
HttpContext  
WindowsIdentity  
Thread  
Name provided by user  
MACHINE\ASPNET  
Name provided by user

**Table 4: IIS integrated Windows**

**Web.config Settings Variable Location Resultant Identity**

<identity impersonate="true"/>  
<authentication mode="Windows" />  
HttpContext  
WindowsIdentity  
Thread  
Domain\UserName

Domain\UserName  
Domain\UserName  
<identity impersonate="false"/>  
<authentication mode="Windows" />  
HttpContext  
WindowsIdentity  
Thread  
Domain\UserName  
MACHINE\ASPNET  
Domain\UserName  
<identity impersonate="true"/>  
<authentication mode="Forms" />  
HttpContext  
WindowsIdentity  
Thread  
Name provided by user  
Domain\UserName  
Name provided by user  
<identity impersonate="false"/>  
<authentication mode="Forms" />

HttpContext.

WindowsIdentity

Thread

Name provided by user

MACHINE\ASPNET

Name provided by user

### **HttpContext.User Property**

Gets or sets security information for the current HTTP request.

**Public Property User As IPrincipal**

### **Property Value**

Security information for the current HTTP request.

### **Remarks**

Setting this property requires the **ControlPrincipal** flag to be set in Flags.

The **HttpContext.User** property provides programmatic access to the properties and methods of the **IPrincipal**

interface. Because ASP.NET pages contain a default reference to the **System.Web** namespace (which contains

the **HttpContext** class), you can reference the members of **HttpContext** on an .aspx page without the fully

qualified class reference to **HttpContext**. For example, you can use just

**User.Identity.Name** to get the

**name of the user on whose behalf the current process is running**. If you want to use the members of

**IPrincipal** from an ASP.NET code-behind module, however, you must include a reference to the **System.Web**

namespace in the module and also fully qualify the reference to the currently active request/response context and the class in **System.Web** you want to use. For example, in a code-behind page you must specify the full name `HttpContext.Current.User.Identity.Name`.

**QUESTION NO: 57**

**You are an application developer for Testkingprep.in. You are developing a method for a Windows Forms application. The method will be used to access local files. The files are located in a folder and its subfolders on drive C of the client computer. All the computers in Testkingprep use the NTFS file system exclusively.**

**The design document for the application specifies the path location for the folder that contains the files. The design document also specifies that the path to the files will be provided as an input parameter to the method. The parameter will be a string parameter.**

**You need to prevent users of the application from accessing any files that are not contained in the specified folder.**

**Which two actions should you include in the method? (Each correct answer presents part of the solution. Choose two)**

- A. Reject any path string that includes a tilde (~).
- B. Reject any path string that includes a folder location that does not match the specified folder location.
- C. Reject any path string that includes either a colon (:) or a backslash (\).
- D. Reject any path string that includes either a \.\ sequence or a \\ sequence.

**Answer: B, D**

**Explanation**

Directory traversal can be accomplished using the '~' and the '\.\' , the '\\' can be used to create a connection to any available sharename. Checking for the valid path string can also be accomplished using regular expressions and simply using the pathname. **Canonicalization** checks must also be made.

**BOLD indicates what the command resolved to:**

C:\>cd \.\windows

**C:\WINDOWS**>cd progra~1

The system cannot find the path specified.

**C:\PROGRA~1**>cd \\windows

'\\windows'

CMD does not support UNC paths as current directories.

C:\PROGRA~1>**cd** \.\..system32

The system cannot find the path specified.

C:\PROGRA~1>**cd** \.\..system32

The system cannot find the path specified.

C:\PROGRA~1>**cd** \.\..windows\system32

```
C:\WINDOWS\system32>cd \\
```

```
\\
```

CMD does not support UNC paths as current directories.

```
C:\WINDOWS\system32>cd \\.\wutemp
```

```
\\.\wutemp'
```

CMD does not support UNC paths as current directories.

```
C:\WINDOWS\system32> cd progra~1 : \.\micro~1
```

```
C:\WINDOWS\system32\MICROS~1> cd \.\progra~1 : \.\micro~1
```

The system cannot find the path specified.

```
C:\WINDOWS\system32>cd \.\
```

```
C:\>cd \.\.\windows\system32
```

```
C:\WINDOWS\system32>cd\
```

```
C:\>cd \.\wind~1\system32
```

The system cannot find the path specified.

```
C:\>cd windows\system32
```

The system cannot find the path specified.

### **Directory Traversal**

HTTP exploits use the Web server software to perform malicious activities. Directory traversal is one such

exploit which lets attackers access restricted directories, execute commands and view data outside the normal

Web server directory where the application content is stored.

### **Detailed description**

Attackers use directory traversal attacks to try to access restricted Web server files residing outside of the Web server's root directory.

The basic role of Web servers is to serve files. Files can be static, such as image and HTML files, or dynamic,

such as ASP and JSP files. When the browser requests a dynamic file, the Web server first executes the file and

then returns the result to the browser. Hence, dynamic files are actually files executed on the Web server.

To prevent users from accessing unauthorized files on the Web server, Web servers provide two main security

mechanisms: the root directory and access controls lists. The root directory limits users' access to a specific

directory in the Web server's file system. All files placed in the root directory and in its sub-directories are

accessible to users. To limit users' access to specific files within the root directory, administrators use access

control lists. Using access control lists, administrators can determine whether a file can be viewed or executed

by users, as well as other access rights.

The root directory prevents attackers from executing files such as cmd.exe on Windows platforms or accessing

sensitive files such as the "passwd" password file on Unix platforms, as these files reside outside of the root directory. The Web server is responsible for enforcing the root directory restriction. By exploiting directory traversal vulnerabilities, attackers step out of the root directory and access files in other directories. As a result, attackers might view restricted files or execute powerful commands on the Web server, leading to a full compromise of the Web server.

A directory traversal vulnerability can exist either in the commercial Web server itself or in the Web application code executed on the Web server. In the case of Web application code, dynamic pages usually receive input from browsers. Here is an example of such an HTTP request:  
`http://www.acme-hackme.com/online/getnews.asp?item=20March2003.html`

In this example, the dynamic page requested by the browser is called `getnews.asp` and the browser sends the Web server the parameter `item` with a value of `20March2003.html`. When executed by the Web server, `getnews.asp` retrieves the file `20March2003.html` from the Web server's file system, renders it and sends it back to the browser which presents it to the user. A skilled attacker will immediately identify the potential problem in this request as the value of the parameter ends with a file extension, in this case "html".

The attacker will then assume that the dynamic page retrieves the file from the file system and uses it. By sending the following URL to the Web server:  
`http://www.acme-hackme.com/online/getnews.asp?item=../../../../WINNT/win.ini`

the attacker causes `getnews.asp` to retrieve the file `../../../../WINNT/win.ini` from the file system and send it to the attacker's browser. The term `"../"` stands for "one directory up". This is a common operating system directive. Therefore, the string `../../../../WINNT/win.ini` means "go four directories up and retrieve the file `win.ini` from there". The attacker needs to guess how many directories to climb in order to get to the desired directory. (In this example the attacker tries to get to `"C:\"` and is assuming that the Web server's root directory is located four directories below `"C:\"`). Guessing the exact combination is very easy. The attacker simply sends multiple requests until the desired result is achieved. The directory traversal vulnerability occurs when programmers fail to validate input received from browsers. In the above example, the `getnews.asp` code does not validate that the value of the `item` parameter does not exceed from the root directory. The directory traversal vulnerability actually bypasses the Web server's root

directory restriction by introducing bad code into the Web server.

Web applications are not the only source of directory traversal vulnerabilities in your Web site. Some

vulnerabilities exist within the Web server. These vulnerabilities can be part of sample files (e.g., sample ASP

files) that exist on the Web server, or can be incorporated into the Web server software.

For example, some

earlier versions of the Microsoft IIS Web server included directory traversal

vulnerabilities that allow attackers

to fully compromise the Web server by executing files on the server. For example, the

following URL:

`http://www.acme-hackme.com/scripts/..%5c../winnt/system32/`

`cmd.exe?/c+dir+c:\`

would execute the `cmd.exe` file (operating system shell) and run the `"dir c:\"` command which lists all files in the

`C:\` directory. Notice the string `"%5c"` that appears in the URL. This is a Web server

escape code. Escape codes

are used to represent normal characters in the form of `%nn`, where `nn` stands for a two-digit number. The escape

code `"%5c"` represents the character `"\"`. The problem is that the IIS root directory

enforcer did not check for

escape codes and allowed that request to execute. The Web server's operating system

understands escape codes

and executes the command.

Escape codes are also very useful for bypassing poorly written filters enforced on input received from users. If

the filter looks for `"../"`, then the attacker could easily change the input to `"%2e%2e/"`.

This has the same

meaning as `"../"`, but is not detected by the filter. The escape code `%2e` represents the character `"."` (dot).

What makes Web-based canonicalization issues so prevalent and hard to defend against is the number of ways

you can represent any character. For example, any character can be represented in a URL or a Web page by

using one or more of the following mechanisms:

- The "normal" 7-bit or 8-bit character representation, also called US-ASCII
- Hexadecimal escape codes
- UTF-8 variable-width encoding
- UCS-2 Unicode encoding
- Double encoding
- HTML escape codes (Web pages, not URLs)

*7-Bit and 8-Bit ASCII*

I trust you understand the 7-bit and 8-bit ASCII representations, which have been used in computer systems for many years.

*Hexadecimal Escape Codes*

Hex escapes are a way to represent a possibly nonprintable character by using its hexadecimal equivalent. For example, the space character is %20, and the pounds sterling character (£) is %A3. You can use this mapping in a URL such as `http://www.northwindtraders.com/my%20document.doc`, which will open `my document.doc` on the Northwind Traders Web site; `http://www.northwindtraders.com/my%20document%2Edoc` will do likewise.

In Chapter 8, I mentioned a canonicalization bug in eEye's SecureIIS tool. The tool looked for certain words in the client request and rejected the request if any of the words were found in the request. However, an attacker could hex escape any of the characters in the request, and the tool would not reject the requests, essentially bypassing the security mechanisms.

#### *UTF-8 Variable-Width Encoding*

Eight-bit Unicode Transformation Format, UTF-8, as defined in RFC 2279 ([www.ietf.org/rfc/rfc2279.txt](http://www.ietf.org/rfc/rfc2279.txt)), is a way to encode characters by using one or more bytes. The variable-byte sizes allow UTF-8 to encode many different byte-size character sets, such as 2-byte Unicode (UCS-2), 4-byte Unicode (UCS-4), and ASCII, to name but a few. However, the fact that one character can potentially map to multiple-byte representations is problematic.

#### **How UTF-8 Encodes Data**

UTF-8 can encode  $n$ -byte characters into different byte sequences, depending on the value of the original characters. For example, a character in the 7-bit ASCII range 0x00-0x7F encodes to **0**7654321, where **0** is the leading bit, set to 0, and 7654321 represents the 7 bits that make up the 7-bit ASCII character. For instance, the letter *H*, which is 0x48 in hex, or 1001000 in binary, becomes the UTF-8 character **0**1001000, or 0x48. As you can see, 7-bit ASCII characters are unchanged by UTF-8. Things become a little more complex as you start mapping characters beyond the 7-bit ASCII range, all the way up to the top of the Unicode range, 0x7FFFFFFF. For example, any character in the range 0x80-0x7FF encodes to **1**10xxxxx **1**0xxxxxx, where **1**10 and **1**0 are predefined bits and each *x* represents one bit from the character. For example, pounds sterling is 0xA3, which is 10100011 in binary. The UTF-8 representation is **1**1000101 **1**0000011, or 0xC5 0x83. However, it doesn't stop there. UTF-8 can encode larger byte-size characters. Table 12-2 outlines the mappings.

**Table 12-2 UTF-8 Character Mappings**

**Character Range Encoded Bytes**

0x00000000- 0x0000007F **0xxxxxxx**

0x00000080- 0x000007FF **110xxxxx 10xxxxxx**

0x00000800- 0x0000FFFF **1110xxxx 10xxxxxx 10xxxxxx**

0x00010000- 0x001FFFFF **11110xxx 10xxxxxx 10xxxxxx 10xxxxxx**

0x00200000- 0x03FFFFFF **111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx**

0x04000000- 0x7FFFFFFF **1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx, 10xxxxxx**

And this is where the fun starts; it is possible to represent a character by using any of these mappings, even

though the UTF-8 specification warns against doing so. All UTF-8 characters should be represented in the

shortest possible format. For example, the only valid UTF-8 representation of the ? character is 0x3F, or

00111111 in binary. On the other hand, an attacker might try using illegal nonshortest formats, such as these:

- 0xC0 0xBF
- 0xE0 0x80 0xBF
- 0xF0 0x80 0x80 0xBF
- 0xF8 0x80 0x80 0x80 0xBF
- 0xFC 0x80 0x80 0x80 0x80 0xBF

A bad UTF-8 parser might determine that all of these formats are the same, when, in fact, only 0x3F is valid.

Perhaps the most famous UTF-8 attack was against unpatched Microsoft Internet Information Server (IIS) 4 and

IIS 5 servers. If an attacker made a request that looked like this—

<http://servername/scripts/../../%c0%af../winnt/system32/cmd.exe>—the server didn't correctly handle %c0%af in

the URL. What do you think %c0%af means? It's 11000000 10101111 in binary; and if it's broken up using the

UTF-8 mapping rules in Table 12-2, we get this: **11000000 10101111**. Therefore, the character is 00000101111,

or 0x2F, the slash (/) character! The %c0%af is an invalid UTF-8 representation of the / character. Such an

invalid UTF-8 escape is often referred to as an *overlong sequence*.

So when the attacker requested the tainted URL, he accessed

<http://servername/scripts/../../winnt/system32/cmd.exe>. In other words, he walked out of the script's virtual

directory, which is marked to allow program execution, up to the root and down into the system32 directory,

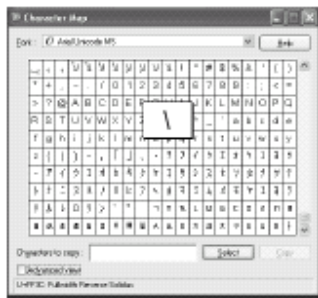
where he could pass commands to the command shell, Cmd.exe.

**MORE INFO**

You can read more about the "File Permission Canonicalization" vulnerability at [www.microsoft.com/technet/security/bulletin/MS00-057.asp](http://www.microsoft.com/technet/security/bulletin/MS00-057.asp).

B. UCS-2 Unicode Encoding

UCS-2 issues are a variation of hex encoding and, to some extent, UTF-8 encoding. Two-byte Universal Character Set, UCS-2, can be hex-encoded in a similar manner as ASCII characters but with the %uNNNN format, where NNNN is the hexadecimal value of the Unicode character. For example, %5C is the ASCII and UTF-8 hex escape for the backslash (\) character, and %u005C is the same character in two-byte Unicode. To really confuse things, %u005C can also be represented by a wide Unicode equivalent called a *fullwidth* version. The fullwidth encodings are provided by Unicode to support conversions between some legacy Asian double-byte encoding systems. The characters in the range %uFF00 to %uFFEF are reserved as the fullwidth equivalents of %20 to %7E. For example, the \ character is %u005C and %uFF3C. You can view these characters by using the Character Map application included with Microsoft Windows. Figure 12-1 shows the backslash character once the Arial Unicode MS font is installed from Microsoft Office



XP.

**Figure 12-1** Using the Character Map application to view Unicode characters.

### C. Double Encoding

Just when you thought you understood the various encoding schemes—and we've looked at only the most common—along comes double encoding, which involves reencoding the encoded data. For example, the UTF-8 escape for the backslash character is %5C, which is made up of three characters—%, 5, and C—all of which can be reencoded using their UTF-8 escapes, %25, %35, and %63. Table 12-3 outlines some double-encoding variations of the \ character.

**Table 12-3** Sample Double Escaping Representations of \

Escape Comments
%5C Normal UTF-8 escape of the backslash character
%255C %25, the escape for % followed by 5C
%%35%63 The % character followed by %35, the escape for 5, and %63, the escape for C
%25%35%63 The individual escapes for %, 5, and C

The vulnerability lies in the mistaken belief that a simple unescape operation will yield clean, raw data. The application then makes a security decision based on the data, but the data might not be fully unescaped.

#### *HTML Escape Codes*

HTML pages can also escape characters by using special characters. For example, angle brackets (< and >) can be represented as &lt; and &gt;, and the pounds sterling symbol can be represented as &pound;. But wait, there's more! These escape sequences can also be represented using the decimal or hexadecimal character values, not just easy-to-remember mnemonics, such as &lt; (less than) and &gt; (greater than). For example, &lt; is the same as &#x3C; (hexadecimal value of the < character) and is also the same as &#60; (decimal value of the < character). A complete list of these entities is available at [www.w3.org/TR/REC-html40/sgml/entities.html](http://www.w3.org/TR/REC-html40/sgml/entities.html).

As you can see, many ways exist to encode data on the Web, which makes making decisions based on the name of a resource a dangerous programming practice. Let's now focus on remedies for these issues.

#### *Web-Based Canonicalization Remedies*

Like all potential canonicalization vulnerabilities, the first defense is simply not to make decisions based on the name of a resource if it's possible to represent the resource name in more than one way.

#### **Restrict What Is Valid Input**

The next best remedy is to restrict what is a valid user request. You created the resources being protected, so you can define the valid ways to access that data and reject all other requests. This is achieved using regular expressions, which are discussed in Chapter 8. Learning to define and use good regular expressions is critical to the security of your application. I'll say it just one more time: always determine what is valid input and reject all other input. It's safer to have a client complain that something doesn't work because of an over-zealous regular expression, than have the service not work because it's been hacked!

#### **Be Careful When Dealing with UTF-8**

If you must manipulate UTF-8 characters, you need to reduce the data to its canonical form by using the *MultiByteToWideChar* function in Windows. The following sample code shows how you can call this function with various valid and invalid UTF-8 characters. You can find the complete code listing on the companion CD in the folder Secureco\Chapter 12\UTF8. Also note that if you want to create UTF-8 characters, you can use

*WideCharToMultiByte* by setting the code page to *CP\_UTF8*.

```
void FromUTF8(LPBYTE pUTF8, DWORD cbUTF8) {
 WCHAR wszResult[MAX_CHAR+1];
 DWORD dwResult = MAX_CHAR;
 int iRes = MultiByteToWideChar(CP_UTF8,
 0,
 (LPCSTR)pUTF8,
 cbUTF8,
 wszResult,
 dwResult);
 if (iRes == 0) {
 DWORD dwErr = GetLastError();
 printf("MultiByteToWideChar() failed - > %d\n", dwErr);
 } else {
 printf("MultiByteToWideChar() returned "
 "%s (%d) wide characters\n",
 wszResult,
 iRes);
 }
}
§
void main() {
 // Get Unicode for 0x5c; should be '\'.
 BYTE pUTF8_1[] = {0x5C};
 DWORD cbUTF8_1 = sizeof pUTF8_1;
 FromUTF8(pUTF8_1, cbUTF8_1);
 // Get Unicode for 0xC0 0xAF.
 // Should fail because this is
 // an overlong '\'.
 BYTE pUTF8_2[] = {0xC0, 0xAF};
 DWORD cbUTF8_2 = sizeof pUTF8_2;
 FromUTF8(pUTF8_2, cbUTF8_2);
 // Get Unicode for 0xC2 0xA9; should be
 // a '©' symbol.
 BYTE pUTF8_3[] = {0xC2, 0xA9};
 DWORD cbUTF8_3 = sizeof pUTF8_3;
 FromUTF8(pUTF8_3, cbUTF8_3);
}
```

### **Design "Parent Paths" Out of Your Application**

Another canonicalization issue relates to the handling of parent paths (..), which can lead to directory traversal

issues if not done correctly. You should design your Web-based system in such a way that parent paths are not

required when data within the application is being accessed. It's common to see a Web application with a

directory structure that requires the use of parent paths, thereby encouraging attackers to attempt to access data outside of the Web root by using URLs like `http://servername/../../boot.ini` to access the boot configuration file, boot.ini. Take a look at the example directory structure in Figure 12-2.

**Figure 12-2** A common Web application directory structure.

As you can see, a common source of images is used throughout the application. To access an image file from a directory that is below the images directory or that is a peer of the images directory—for example, advertising and private—your application will need to move out of the current directory into the images directory, therefore requiring that your application use parent paths. For example, to load an image, a file named

`/private/default.aspx` would need to use `<IMG>` tags that look like this:

```

```

However, in Windows 2000 and later, the need for parent paths can be reduced. You can create a junction point

to the images directory or a hard link to an individual file in the images directory from within the present

directory. Figure 12-3 shows what the newer directory structure looks like. It's more secure because there's no

need to access any file or directory by using parent paths; your application can remove multiple dots as a

requirement in a valid file request.

**Figure 12-3** A common Web application directory structure using links to a parent or peer directory.

With this directory format in place, the application can access the image without using parent paths, like so:

```

```

You can create junction points by using the `Linkd.exe` tool included in the Windows 2000 Resource Kit, and

you can link to an individual file by using the `CreateHardLink` function. The following is a simple example of

using the `CreateHardLink` function to create hard links to files. You can also find this example code on the

companion CD in the folder `Secureco\Chapter 12\HardLink`.

```
/*
```

```
HardLink.cpp
```

```
*/
```

```
#include "stdafx.h"
```

```
DWORD DoHardLink(LPCSTR szName, LPCSTR szTarget) {
```

```
DWORD dwErr = 0;
```

```
if (!CreateHardLink(szName, szTarget, NULL))
```

```
dwErr = GetLastError();
```

```
return dwErr;
```

```

}
void main(int argc, char* argv[]) {
if (argc != 3) {
printf("Usage: HardLink <linkname> <target>\n");
}
DWORD dwErr = DoHardLink(argv[1], argv[2]);
if (dwErr)
printf("Error calling CreateHardLink() - > %d\n", dwErr);
else
printf("Hard link created to %s\n", argv[2]);
}

```

#### **NOTE**

Just say no to parent paths. If you remove the requirement for parent paths in your application, anyone attempting to access a resource by using parent paths is, by definition, an attacker!

#### **QUESTION NO: 58**

**You are an application developer for Testkingprep.in. Part of the application that you are developing accepts user input from a TextBox control. The information entered by the user must be alphanumeric only, and it must contain no symbols or punctuation. You need ensure that the user's input contains only the appropriate data before using the input elsewhere in the application. Your solution must not require users of the application to take additional steps when entering data. What should you do?**

- A. Modify the TextChanged event handler of the TextBox control so that the Text property of the text box is cleared whenever a non-alphanumeric character is detected.
- B. Use the following regular expression to modify the user's input.  
[^\w\.\@-]
- C. Store the user's input in a variable named userInput.  
Use the following expression to modify the user's input.  
userInput.Replace("@-] ", "")
- D. Convert the user's input to all lowercase characters.

**Answer: B**

#### **Explanation**

Never Trust User Input!

I know this injunction sounds harsh, as if people are out to get you. But many are. If you accept input from users, either directly or indirectly, it is imperative that you validate the input before using it, because people will try to make your application fail by tweaking the input to represent invalid data. The first golden rule of user

input is, *All input is bad until proven otherwise*. Typically, the moment you forget this rule is the moment you are attacked. In this section, we'll focus on the many ways developers read input, how developers use the input, and how attackers try to trip up your application by manipulating the input. Let me introduce you to the second golden rule of user input: *Data must be validated as it crosses the boundary between untrusted and trusted environments*. By definition, trusted data is data you or an entity you explicitly trust has complete control over; untrusted data refers to everything else. In short, any data submitted by a user is initially untrusted data. The reason I bring this up is many developers balk at checking input because they are positive that the data is checked by some other function that eventually calls their application, and they don't want to take the performance hit of validating the data. But what happens if the input comes from a source that is not checked, or the code you depend on is changed because it assumes some other code performs a validity check?

#### **NOTE**

A somewhat related question is, what happens if an honest user simply makes an input mistake that causes your application to fail? Keep this in mind when I discuss some potential vulnerabilities and exploits.

I once reviewed a security product that had a security flaw because a small chance existed that invalid user input would cause a buffer overrun and stop the product's Web service. The development team claimed that it could not check all the input because of potential performance problems. On closer examination, I found that not only was the application a critical network component—and hence the potential damage from an exploit was immense—but also it performed many time-intensive and CPU-intensive operations, including public-key encryption, heavy disk I/O, and authentication. I doubted much that a half dozen lines of input-checking code would lead to a performance problem. As it turned out, the code did indeed cause no performance problems, and the code was rectified.

#### *User Input Remedies*

As with all user input issues, the first rule is to determine which input is valid and to reject all other input. (Have I said that enough times?) Other not-so-paranoid options exist and offer more functionality with potentially less

security. I'll discuss some of these also.

### **A Simple and Safe Approach: Be Hardcore About Valid Input**

In the cases of the Web-based form and SQL examples earlier, the valid characters for a username can be easily restricted to a small set of valid characters, such as A-Za-z0-9. The following server-side JScript snippet shows

how to construct and use a regular expression to parse the username at the server:

```
// Determine whether username is valid.
// Valid format is 1 to 32 alphanumeric characters.
var reg = /^[A-Za-z0-9]{1,32}$/g;
if (reg.test(Request.form("name"))) > 0) {
 // Cool! Username is valid.
} else {
 // Not cool! Username is invalid.
}
```

### **A Regular Expression Rosetta Stone**

Regular expressions are incredibly powerful, and their usefulness extends beyond just restricting input. They constitute a technology worth understanding for solving many complex data manipulation problems. I write

many applications, mostly in Perl and C#, that use regular expressions to analyze log files for attack signatures

and to analyze source code for security defects.

### ***Regular Expressions in Managed Code***

Most if not all applications written in C#, Managed C++, Microsoft Visual Basic .NET, ASP.NET, and so on

have access to the .NET Framework and as such can use the

*System.Text.RegularExpressions* namespace. I've

already outlined its syntax earlier in this chapter. However, for completeness, following are C#, Visual Basic

.NET, and Managed C++ examples of the date extraction code I showed earlier in Perl.

### **C# Example**

```
// C# Example
String s = @"We leave at 12:15pm for Mount Doom.";
Regex r = new Regex(@".*(\d{2}:\d{2}[ap]m)", RegexOptions.IgnoreCase);
if (r.Match(s).Success)
 Console.WriteLine(r.Match(s).Result("$1"));
```

### **Visual Basic .NET Example**

```
' Visual Basic .NET example
Imports System.Text.RegularExpressions
.
Dim s As String
Dim r As Regex
s = "We leave at 12:15pm for Mount Doom."
r = New Regex(".*(\d{2}:\d{2}[ap]m)", RegexOptions.IgnoreCase)
If r.Match(s).Success Then
```

```
Console.WriteLine(r.Match(s).Result("$1"))
End If
```

### **Managed C++ Example**

```
// Managed C++ version
#using <mscorlib.dll>
#include <tchar.h>
#using <system.dll>
using namespace System;
using namespace System::Text;
using namespace System::Text::RegularExpressions;

String *s = S"We leave at 12:15pm for Mount Doom.";
Regex *r = new Regex(".*(\\d{2}:\\d{2}[ap]m)", IgnoreCase);
if (r->Match(s)->Success)
Console::WriteLine(r->Match(s)->Result(S"$1"));
```

Note that the same code applies to ASP.NET because ASP.NET is language-neutral.

### *D. Regular Expressions in Script*

The base JavaScript 1.2 language supports regular expressions by using syntax similar to Perl. Netscape

Navigator 4 and later and Microsoft Internet Explorer 4 and later also support regular expressions.

```
var r = /.*(\\d{2}:\\d{2}[ap]m)/;
var s = "We leave at 12:15pm for Mount Doom.";
if (s.match(r))
alert(RegExp.$1);
```

Regular expressions are also available to developers in Microsoft Visual Basic Scripting Edition (VBScript)

version 5 via the *RegExp* object:

```
Set r = new RegExp
r.Pattern = ".*(\\d{2}:\\d{2}[ap]m)"
r.IgnoreCase = True
Set m = r.Execute("We leave at 12:15pm for Mount Doom.")
MsgBox m(0).SubMatches(0)
\\w ----> matches any word character, equivalent to [a-zA-Z0-9]
```

### **Regular Expressions**

Regular expressions are a concise and flexible notation for finding and replacing patterns of text. The regular

expressions used within Visual Studio are a superset of the expressions used in Visual C++ 6.0, with a simplified syntax.

You can use the following regular expressions in the Find, Replace, Find in Files or Replace in Files dialog boxes to refine and expand your search.

**Note** You must select the Use check box in the Find, Replace, Find in Files, and Replace in Files dialog boxes

before using any of the following expressions as part of your search criteria.

The following expressions can be used to match characters or digits in your search string:

### **Expression Syntax Description**

Any character `.` Matches any one character except a line break.

Maximal — zero or more `*` Matches zero or more occurrences of the preceding expression.

Maximal — one or more `+` Matches at least one occurrence of the preceding expression.

Minimal — zero or more `@` Matches zero or more occurrences of the preceding expression, matching as few characters as possible.

Minimal — one or more `#` Matches one or more occurrences of the preceding expression, matching as few characters as possible.

Repeat  $n$  times `^n` Matches  $n$  occurrences of the preceding expression. For example, `[0-9]^4` matches any 4-digit sequence.

Set of characters `[]` Matches any one of the characters within the `[]`. To specify a range of characters, list the starting and ending character separated by a dash (`-`), as in `[a-z]`.

Character not in set `[^...]` Matches any character not in the set of characters following the `^`.

Beginning of line `^` Anchors the match to the beginning of a line.

End of line `$` Anchors the match to the end of a line.

Beginning of word `<` Matches only when a word begins at this point in the text.

End of word `>` Matches only when a word ends at this point in the text.

Grouping `()` Groups a subexpression.

Or `|` Matches the expression before or after the OR symbol `()`. Mostly used within a group. For example, `(sponge|mud) bath` matches "sponge bath" and "mud bath."

Escape `\` Matches the character following the backslash (`\`). This allows you to find characters used in the regular expression notation, such as `{` and `^`. For example, `\^` Searches for the `^` character.

Tagged expression `{ }` Tags the text matched by the enclosed expression.

$n$ th tagged text `\n` In a Find or Replace expression, indicates the text matched by the  $n$ th tagged expression, where  $n$  is a number from 1 to 9.

In a Replace expression, `\0` inserts the entire matched text.

Right-justified field `\(w,n)` In a Replace expression, right-justifies the  $n$ th tagged expression in a field at least  $w$  characters wide.

Left-justified field `\(-w,n)` In a Replace expression, left-justifies the  $n$ th tagged expression in a field at least  $w$  characters wide.

Prevent match `~(X)` Prevents a match when  $X$  appears at this point in the expression. For example, `real~(ity)` matches the "real" in "realty" and "really," but not the "real" in

"reality."

Alphanumeric character :a Matches the expression  
([a-zA-Z0-9]).

Alphabetic character :c Matches the expression  
([a-zA-Z]).

Decimal digit :d Matches the expression  
([0-9]).

Hexadecimal digit :h Matches the expression  
([0-9a-fA-F]+)..

Identifier :i Matches the expression  
([a-zA-Z\_\$][a-zA-Z0-9\_\$]\*).

Rational number :n Matches the expression  
(([0-9]+.[0-9]\*)|([0-9]\*.[0-9]+)|([0-9]+)).

Quoted string :q Matches the expression ((("[^"]\*" )|('[^']\*' ) )

Alphabetic string :w Matches the expression  
([a-zA-Z]+)

Decimal integer :z Matches the expression  
([0-9]+).

Escape \e Unicode U+001B.

Bell \g Unicode U+0007.

Backspace \h Unicode U+0008.

Line break \n Matches a platform-independent line break. In a  
Replace expression, inserts a line break.

Tab \t Matches a tab character, Unicode U+0009.

Unicode character \x#### or \u#### Matches a character given by Unicode value where  
#### is hexadecimal digits. You can specify a character  
outside the Basic Multilingual Plane (that is, a  
surrogate) with the ISO 10646 code point or with two  
Unicode code points giving the values of the surrogate  
pair.

The following table lists the syntax for matching by standard Unicode character  
properties. The two-letter  
abbreviation is the same as listed in the Unicode character properties database. These  
may be specified as part  
of a character set. For example, the expression [:Nd:Nl:No] matches any kind of digit.

### **Expression Syntax Description**

Uppercase letter :Lu Matches any one capital letter. For example, :Luhe  
matches "The" but not "the".

Lowercase letter :Ll Matches any one lower case letter. For example,  
:Llhe matches "the" but not "The".

Title case letter :Lt Matches characters that combine an uppercase letter  
with a lowercase letter, such as Nj and Dz.

Modifier letter :Lm Matches letters or punctuation, such as commas, cross  
accents, and double prime, used to indicate  
modifications to the preceding letter.

Other letter :Lo Matches other letters, such as gothic letter ahsa.

Decimal digit :Nd Matches decimal digits such as 0-9 and their full-width equivalents.

Letter digit :Nl Matches letter digits such as roman numerals and ideographic number zero.

Other digit :No Matches other digits such as old italic number one.

Open punctuation :Ps Matches opening punctuation such as open brackets and braces.

Close punctuation :Pe Matches closing punctuation such as closing brackets and braces.

Initial quote punctuation :Pi Matches initial double quotation marks.

Final quote punctuation :Pf Matches single quotation marks and ending double quotation marks.

Dash punctuation :Pd Matches the dash mark.

Connector punctuation :Pc Matches the underscore or underline mark.

Other punctuation :Po Matches commas (,), ? , " , ! , @ , # , % , & , \* , \ , colons (:), semi-colons (;), ' , and /.

Space separator :Zs Matches blanks.

Line separator :Zl Matches the Unicode character U+2028.

Paragraph separator :Zp Matches the Unicode character U+2029.

Non-spacing mark :Mn Matches non-spacing marks.

Combining mark :Mc Matches combining marks.

Enclosing mark :Me Matches enclosing marks.

Math symbol :Sm Matches + , = , ~ , | , < , and > .

Currency symbol :Sc Matches \$ and other currency symbols.

Modifier symbol :Sk Matches modifier symbols such as circumflex accent, grave accent, and macron.

Other symbol :So Matches other symbols, such as the copyright sign, pilcrow sign, and the degree sign.

Other control :Cc Matches end of line.

Other format :Cf Formatting control character such as the bidirectional control characters.

Surrogate :Cs Matches one half of a surrogate pair.

Other private-use :Co Matches any character from the private-use area.

Other not assigned :Cn Matches characters that do not map to a Unicode character.

In addition to the standard Unicode character properties, the following additional properties may be specified.

These properties may be specified as part of a character set.

### **Expression Syntax Description**

Alpha :Al Matches any one character. For example, :Alhe matches words such as "The", "then", and "reached".

Numeric :Nu Matches any one number or digit.

Punctuation :Pu Matches any one punctuation mark, such as ? , @ , ' , and so on.

White space :Wh Matches all types of white space, including publishing and ideographic spaces.

Bidi :Bi Matches characters from right-to-left scripts such as Arabic and Hebrew.

Hangul :Ha Matches Korean Hangul and combining Jamos.

Hiragana :Hi Matches hiragana characters.

Katakana :Ka Matches katakana characters.

Ideographic/Han/Kanji :Id Matches ideographic characters, such as Han and Kanji.

.NET Framework Regular Expressions

Provides a brief introduction to .NET regular expressions.

Regular Expressions as a Language

Provides an overview of the programming-language aspect of regular expressions.

Regular Expression Classes

Provides detailed information and code examples illustrating how to use the regular expression classes.

Details of Regular Expression Behavior

Provides detailed information about the capabilities and behavior of .NET Framework regular expressions.

Regular Expression Examples

Provides code examples illustrating typical uses of regular expressions.

System.Text.RegularExpressions

Provides class-library reference information for the .NET Framework

**System.Text.RegularExpressions** namespace.

**Regular Expression Validator Control Sample**

The regular expression validator control shown here extends the base validator control described in the Base

Validator Control Sample. This validator adds the following functionality to the base validator:

- It exposes a property named **ValidationExpression** that allows a user (page developer) to specify a regular expression.
- It overrides the **EvaluateIsValid** method (defined as an abstract method in **BaseDomValidator**) to provide logic to determine whether the field to validate matches the pattern specified by the regular expression.
- It overrides **AddAttributesToRender** (inherited from **WebControl**) to provide a client-side handler for the evaluation logic. The client-side handler is a function defined in the script library.

#### QUESTION NO: 59

**You are an application developer for Testkingprep.in. You are conducting a code review of an application that was developed by another developer. The application includes a function named `NeedPermissions()`. This function accepts a Boolean input parameter. If this parameter is set to true, the function demands permissions to access Testkingprep's Active Directory database. If the parameter is set to false, the function releases the permissions.**

**The application contains functions that access Active Directory. Each of these functions is written by using the following code template.**

```
Private type function_name() { try {
 NeedPermission(true);
 //Code to access Active Directory is here
}
catch(exception e) {
 //Code to log exceptions is here
 //Exit the function
```

**You need to improve the security of this code template while maintaining its functionality. You decide to replace the existing code template.**

**Which code template should you use?**

```
A. private type function_name() {
 try {
 NeedPermission(true);
 //Code to access Active Directory is here
 }
 catch(Exception e) {
 //Code to log exceptions is here
 }
 finally {
 NeedPermission(false);
 }
}
```

```
B. private type function name() {
 NeedPermissions(true);
 try {
 //Code to access Active Directory is here
 }
 catch(Exception e) {
 //Code to log exceptions is here
 //Exit the function
 }
 NeedPermissions(false);
}
```

```
C. private type function_name() {
 try {
 NeedPermission(true)
 //Code to access Active Directory is here
 }
 catch(Exception e) {
 //Code to log exceptions is here
 NeedPermissions(false);
 //Exit the function
```

```

}
}
D. private type function_name() {
try {
NeedPermissions(true);
//Code to access Active Directory is here
}

```

**Answer: A**

### **Explanation**

#### **try-catch-finally**

A common usage of **catch** and **finally** together is to obtain and use resources in a **try** block, deal with exceptional circumstances in a **catch** block, and release the resources in the **finally** block.

#### **Example**

```

// try-catch-finally
using System;
public class EHClass
{
 public static void Main ()
 {

 try
 {
 Console.WriteLine("Executing the try statement.");
 throw new NullReferenceException();
 }
 catch(NullReferenceException e)
 {
 Console.WriteLine("{0} Caught exception #1.", e);
 }
 catch
 {
 Console.WriteLine("Caught exception #2.");
 }

 finally
 {
 Console.WriteLine("Executing finally block.");
 }
 }
}

```

#### **Output**

Executing the try statement.

System.NullReferenceException: Attempted to dereference a null object reference.  
at EHClass.Main() Caught exception #1.

Executing finally block.

For more information and examples on rethrowing exceptions, see try-catch, 8.10 The try statement and

Throwing Exceptions.

### See Also

C# Keywords | Compare to C++ | Exception Handling Statements | throw | Throwing Exceptions | C. Grammar

### Parts

*tryStatements*

Optional. Statement(s) where an error can occur. Can be a compound statement.

### Catch

Optional. Multiple **Catch** blocks permitted. If an exception occurs while processing the **Try** block, each

**Catch** statement is examined in textual order to determine if it handles the exception.

*Exception*

represents the exception that has been thrown.

*exception*

Optional. Any variable name. The initial value of *exception* is the value of the thrown error. Used with

**Catch** to specify the error caught.

*type*

Optional. Specifies the type of class filter. If the value of *exception* is of the type specified by *type* or of

a derived type, the identifier becomes bound to the exception object.

### When

Optional. A **Catch** statement with a **When** clause will only catch exceptions when *expression* evaluates

to **True**. A **When** clause is only applied after checking the type of the exception, and *expression* may

refer to the identifier representing the exception.

*expression*

Optional. Must be implicitly convertible to **Boolean**. Any expression that describes a generic filter.

Typically used to filter by error number. Used with **When** keyword to specify circumstances under which the error is caught.

*catchStatements*

Optional. Statement(s) to handle errors occurring in the associated **Try** block. Can be a compound statement.

### Exit Try

Optional. Keyword that breaks out of the **Try...Catch...Finally** structure. Execution resumes with the

**Finally** block if present, otherwise with the code immediately following the **End Try** statement. Not allowed in **Finally** blocks.

#### **Finally**

Optional. A **Finally** block is always executed when execution leaves any part of the **Try** statement.

#### *finallyStatements*

Optional. Statement(s) that are executed after all other error processing has occurred.

#### **End Try**

Terminates the **Try...Catch...Finally** structure.

#### **Remarks**

Local variables from a **Try** block are not available in a **Catch** block because they are separate blocks. If you

want to use a variable in more than one block, declare the variable outside the **Try...Catch...Finally** structure.

If errors occur that the programmer has not handled, Visual Studio for Applications simply provides its normal error message to a user, as if there was no error handling.

The **Try** block contains code where an error can occur, while the **Catch** block contains code to handle any error

that does occur. If an error occurs in the **Try** block, program control is passed to the appropriate **Catch**

statement for disposition. The *exception* argument is an instance of the **Exception** class or an instance of a class

that derives from the **Exception** class corresponding to the error that occurred in the **Try** block. The **Exception**

class instance contains information about the error including, among other things, its number and message.

In partial trust situations, such as an application hosted on a network share,

**Try...Catch...Finally** will not catch

security exceptions that occur before the method containing the call is invoked. The following example, when

placed on a server share and run from there, for example, will produce the error: "Sub System.Security.SecurityException: Request Failed. For more information on Security Exceptions, see

SecurityException Class.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button1.Click
```

```
Try
```

```
Process.Start("http://www.microsoft.com")
```

```
Catch ex As Exception
```

```
MsgBox("Can't load Web page" & vbCrLf & ex.Message)
```

```
End Try
```

```
End Sub
```

In such a partial trust situation, you would need to put the `Process.Start` statement in a separate **Sub**. The initial

call to the **Sub** will fail, allowing **Try...Catch** to catch it before the Sub containing **Process.Start** is launched and the security exception produced.

**Note** If a **Try** statement does not contain at least one **Catch** block it must contain a **Finally** block.

#### **QUESTION NO: 60**

**You are an application developer for Testkingprep.in. You are conducting a code review of an application that was developed by another developer. The code declares a variable named permvalue and a variable named grouplist.**

**A portion of the application code defines security permissions for the user. The application is designed so that permvalue contains an integer that indicates various permissions within the application, and grouplist contains the name of a user group. The permvalue variable also contains values that indicate other information about the user. The grouplist and permvalue variables are initially populated by other components, which are called by the main application.**

**The application contains the following code segment. (Line numbers are included for reference only.)**

```
01 switch(grouplist) [
02 case "Admin";
03 case "Administrator":
04 permvalue = permvalue | 256;
05 break;
06 case "Reviewer":
07 permvalue = permvalue | 138;
08 break;
09 case "Manager":
10 permvalue = permvalue | 64;
11 break;
12 }
```

**The design document for the application states that permvalue must have a value of zero when the user has no permissions. The design document also states that users not belonging to one of the four predefined groups must have no permissions.**

**You need to ensure that the code segment assigns the correct value to permvalue in all circumstances.**

**What should you do?**

A. Add the following code before line 01 of the code segment.

```
if(permvalue == 0) {
 throw new ApplicationException();
}
```

B. Add the following code before line 01 of the code segment.

```
permvalue = 0;
```

C. Add the following code between lines 01 and 02 of the code segment.

```
case "":
 permvalue = 0;
```

break;

D. Add the following code between lines 11 and 12 of the code segment.

default:

permvalue = 0;

**Answer: D**

### **Explanation**

Since the string variable and the integer variable can contain values populated from other components, when the 'Select Case' is reached their contents will have those values as the default. Assigning '0' to the permissions at the beginning of the 'Select Case' would mean the value of the permission would always be zero before reaching the Select Case statements. Assigning '0' as the last case would catch all that are not part of the Select Case specification.

### **QUESTION NO: 61**

**You are an application developer for Testkingprep.in. You are conducting a code review of an assembly written by another developer. The assembly is named MyAssembly.exe. The assembly is for an application that accesses data in a Microsoft SQL Server database. All users of the application have access to the database by using their Microsoft Windows user accounts.**

**The assembly contains the following code segment.**

```
string userid;
string password;
userid = "sa";
password = "";
SqlConnection sqlConnection = new SqlConnection();
string connectionString;
connectionString = "data source=myServer";
connectionString += ";initial catalog=myDatabase";
connectionString += ";user id=" + userid;
connectionString += ";password=" + password;
sqlConnection.ConnectionString = connectionString;
sqlConnection.Open();
```

**You need to improve the security of the code segment.**

**What should you do?**

A. Replace the code segment with the following code segment.

```
SqlConnection sqlConnection = new SqlConnection();
string connectionString;
connectionString = "data source=myServer";
connectionString += ";Integrated Security=SSPI";
connectionString += "initial catalog=myDatabase";
sqlConnection.ConnectionString = connectionString;
```

```
sqlConnection.Open();
```

B. Replace the code segment with the following code segment.

```
SqlConnection sqlConnection = new SqlConnection();
```

```
string connectionString;
```

```
connectionString =
```

```
“data source=myserver;initial catalog=myDatabase;user id=sa;password
```

```
sqlConnection.ConnectionString = connectionString;
```

```
sqlConnection.Open();
```

C. Run the **caspol.exe –resolvperm MyAssembly.exe** command from the command line.

D. Run the **permview /decl MyAssembly.exe** command from the command line.

**Answer: A**

### **Explanation**

Never use the SQL default administrative account ‘SA’ and a blank password “”, for any sort of access. This

account has all access to all databases regardless of who or what created it as well as can be used to take

complete control of the machine and even the network. SQL has hundreds of extended stored procedures

(XP\_???) of them xp\_cmd can be use to elevate permissions well beyond what is needed and be used to

compromise almost every aspect of the system and the network.

### **Security Recommended Practices**

Microsoft recommends the following practices to help you protect your data and applications from malicious

users and accidental user actions.

Notification Services Security Practices

- Run the NS\$instance\_name service under a weak domain or local account. Do not use the LocalSystem

or NetworkService service account or any account in the **Administrators** group.

However, if you are using a delivery protocol that requires the account that the service runs under to have

additional privileges, you must use higher privileges. For example, sending notifications using an Internet

Information Services (IIS) SMTP server requires the account under which the service runs to be a member

of the local Administrators group.

- Ensure that the password used by the service account is a strong password. For more information about

strong passwords, see "Creating Strong Passwords" in the Microsoft Windows documentation.

- Ensure that all code run by the NS\$instance\_name service, such as custom event providers, content

formatters, and protocols, is from a trusted source. Notification Services assumes that code listed in the

application definition file (ADF) comes from a trusted source.

- Secure all folders containing configuration files or application data. For more information about

securing files and folders, see File and Folder Security.

SQL Server Security Practices

- **When installing SQL Server, never allow a blank sa password, even if you select the integrated security mode. This guarantees that if the security mode changes to mixed mode, the sa account will still have a password.**

- Use Windows Authentication whenever possible. Windows Authentication provides advanced security

features, such as policies for password length, complexity, and expiration. Note that if the NS\$instance\_name service uses a SQL Server user name and password to connect to SQL Server, this user

name and password are encrypted and stored in the registry.

- If you use SQL Server Authentication, use strong passwords for the SQL Server login accounts and change the passwords periodically.

- Do not grant unnecessary permissions to the **public** role in each database. The **public** role is a special database role to which every database user belongs, and cannot be dropped from the database. Notification

Services does not use the **public** role.

- Do not grant database access to the **guest** user account. The **guest** user account allows a SQL Server login account that does not have a database user account to access a database.

- Consider encrypting the database files using NTFS file encryption. This can decrease performance, so you must weigh optimal performance against file security.

Network Communications Security Practices

- To reduce the possibility of intruders viewing data as it is being transferred between Notification

Services and the database, use encrypted communication between client applications and SQL Server. For

more information, see "Using Encryption Methods" in SQL Server Books Online.

- If you are using an HTTP protocol to post data to a Web server, and if the Web server supports SSL,

post the notification using an address that starts with https://. This form of address encrypts the data that is

sent to the Web server.

Physical Security Practices

Ensure that your servers are located in an area that is adequately secured. If a malicious user can physically

access the server, the server is not secure.

### **Database Security**

One of the most common scenarios for a distributed application involves reading and writing data on a remote

database. The dilemma that arises is how to do so securely while maintaining application scalability. Where you

choose to manage security in your application will greatly impact, either negatively or positively, the scalability of your application.

To achieve scalability using database connection pooling foregoes having the database manage security. This is because database connection pooling requires the connection string be identical to poolconnections. Therefore, you must manage security elsewhere. If you must track database operations on per user basis, consider adding a parameter for user identity to each operation and manually log user actions in the database.

Following the advice above, another issue is how to store the database connection string, which typically

contains security credentials, so multiple users can access it without compromising security. Most sample applications demonstrate storing the connection string in the Web.config or global.asax files. However, because these files are plain text files that have limited security, it is not the best location for storing this information.

Should an intruder compromise your Web server's security, these files would be easily accessible. Here are just a few alternatives:

- If using the Web.config file, store the connection string encrypted and then decrypt the connection string in your application code when needed.
- Build a COM+ application using the ServicedComponent Class and store the connection string in the construct string for that component.

When storing sensitive information in the constructor string, you should verify the following:

- Only the appropriate users/groups belong to the Reader role of the System Package. However,

you must carefully manage COM+ to prevent it from being unable to read its own configuration.

- You have controlled and audited access to the %windir%\Registration folder, where the COM+ configuration database (RegDB) stores its files.

For more information, see ServicedComponent Class.

- Use integrated security to make a trusted connection with SQL Server. This makes it possible for you to use a connection string that eliminates the need for storing a password in the connection string, such as:

```
"Data Source=mySqlServer;Integrated Security=SSPI;Initial Catalog=myDB"
```

There are some drawbacks to using integrated security, most of which you can overcome. Because

integrated security requires a Windows account, it defeats connection pooling if you impersonate each authenticated principal using an individual Windows account. However, if you instead impersonate a limited number of Windows accounts, with each account representing a particular role, you can overcome this drawback. Each Windows account must be a domain account with IIS and SQL Server in the same or trusted domains. Alternatively, you can create identical (including passwords) Windows accounts on each machine. After a typical installation, the default security authentication mode is Windows Authentication for SQL Server 2000, which is different from SQL Server 7.0. In SQL Server 7.0, the default authentication mode is Mixed (Windows Authentication Mode and SQL Server Authentication). Windows Authentication is a better security method because of the additional security features it provides, such as secure validation and encryption of passwords, password expiration and auditing. For more information, see Authentication Modes.

If you configure SQL Server to use Windows Authentication, you could create one Windows account for read-only operations and another Windows account for read/write operations. You then map each Windows account to a SQL Server login and establish the desired permissions. Using application logic, you then determine which Windows account to impersonate when performing database operations. In SQL Server, you can add any Windows user account as a member of a fixed database role. Each member gains the permissions applied to the fixed database role. For more information, see Managing Permissions.

For SQL Server 7.0, integrated security does not work with SQL Server's TCP/IP network library, but uses the named pipes network library instead.

As an added security measure, the `ConnectionString` property of the `SqlConnection` object does not persist or return the full connection string by default. To do so, you must set `Persist Security Info` to `true`.

#### **QUESTION NO: 62**

**You are an application developer for Testkingprep.in. You are conducting a code review of an application that updates a Microsoft SQL Server database named Payroll. This database is used by other applications. The application contains the following code segment.**

```
public void calculateAndStore(SqlConnection conn,
```

```

string ID, string bonus, string salary) {
 salary=Convert.ToString(Convert.ToDecimal(salary)*1.05m);
 bonus=Convert.ToString(Convert.ToDecimal(salary)*0.05m);
 if (ID.Length==0)
 throw new ApplicationException("Error – Empty ID");
 string newID="ID-" + ID;
 string strUpdate = "UPDATE Payroll SET EmployeeID = " + newID +
 ", Bonus = " + bonus + ", Salary = " + salary + " " +
 "WHERE EmployeeID=" + ID + """;
 SqlCommand cmd=new SqlCommand(strUpdate,conn);
 cmd.Connection.Open();
 cmd.ExecuteNonQuery(); }

```

**The values in the string variables named ID, Bonus, and Salary are contained in the Payroll database.**

**The purpose of the code segment is to calculate new values for ID, Bonus, and Salary, and to update those values in the Payroll database.**

**You need to improve the security of this application.**

**What should you do?**

- A. Validate that the Salary value is within the range for the data type in the SQL Server database.
- B. Validate the contents of the ID value before updating it in the SQL Server database.
- C. Validate the length of the ID value before updating it in the SQL Server database.
- D. Enclose the body of the function within a **try-catch** block.

**Answer: D**

### **Explanation**

There are several things worrisome about this question. Since all input must be considered evil until otherwise proven, even input from and passed by applications, there must be a validation step used at every opportunity.

This will increase the code segment and the application will take a perceived performance hit. However, this is still better than taking the chance that the data being used or passed has not been manipulated. Parameterized stored procedures or parameterized SQL statements should be used instead of dynamic SQL. Since the information is being pulled from the database, we have to assume that earlier processes in the application have handled the scrubbing and validation of the data during the original data entry or it should not be there. However, a less than trustworthy database administrator could still manipulate the data anytime and the database could have been compromised by some other means.

**QUESTION NO: 63**

**You are an application developer for Testkingprep.in. You are developing a client application that queries a Microsoft SQL Server database. The application uses an unmanaged component to retrieve data from another application, and your application uses that data as part of a SQL query. In the application code, you use a variable named externalobject to refer to the unmanaged component. A variable named calcval contains an integer value that is calculated by your application. A SqlCommand object named sqlcmd is already defined and associated with an open ADO.NET connection to the SQL Server database.**

**The application contains the following code segment.**

```
string myquery;
myquery = "INSERT INTO DataStore (ExternalID, CalcValue)";
myquery += " VALUES(" + externalobject.LegacyData + ",";";
myquery += calcval.ToString() + ")";
sqlcmd.CommandText = myquery;
sqlcmd.ExecuteNonQuery();
```

**You need to improve the security of this code segment.**

**What should you do?**

- A. Place the code segment within a **try-catch** block.
- B. In the code segment, ensure that the value of externalobject.LegacyData meets the length and type requirements of the SQL Server table.
- C. Validate the externalobject.LegacyData contains only expected data and no additional SQL statements.
- D. Copy the contents of externalobject.LegacyData into a string variable, and append the string variable to the SQL statement.

**Answer: C**

### **Explanation**

It is best for the approach that you must validate and cleanse data before it is used and/or store.

Rule number two is: *data must be validated as it crosses the boundary between untrusted and trusted*

*environments*. By definition, trusted data is data you or an entity you explicitly trust has complete control over;

untrusted data refers to everything else. In short, any data submitted by a user is initially untrusted data.

When it comes to SQL statements, all dynamic SQL is bad, and parameterized stored procedures must be used.

Dynamic SQL can be easily compromised and used for SQL injection attacks.

All relational databases—including SQL Server, Oracle, IBM DB2, and MySQL—are susceptible to SQL injection

attacks. You can buy products that protect your system from SQL injection, but for most businesses,

the defense against SQL-injection attack must be code-based. The opening for SQL-injection attacks comes primarily through Web applications that combine user input with dynamic SQL to form SQL commands that the application sends to the database. Here are four important steps you can take to protect your Web applications from SQL-injection attacks. In addition to the following tips, the Microsoft Patterns and Practices Library that I highlighted last month provides advice about securing your data-access applications.

#### 4. Principle of Least Privilege

The account an application uses to connect to the database should have only the privileges that application requires. The security permissions that an intruder gains from a compromised application define the harm that the intruder can inflict. Applications shouldn't connect as sa or with the Administrator account. Instead, the account should have permissions to access only the database objects it needs.

#### 3. Validate All Input

If an input field should contain numeric data, then verify that users enter only numbers. If character data is acceptable, check for unexpected characters. Make sure your application looks for characters such as semicolons, equals signs, double dashes, brackets, and SQL keywords. The .NET Framework provides regular expressions that enable complex pattern matching, a good way to test user input. Limiting the length of accepted user input is also a good idea. Validating your input might seem obvious, but many applications are vulnerable to SQL-injection attacks because intruders can use the openings that Web applications offer.

#### 2. Avoid Dynamic SQL

Dynamic SQL is a great tool for performing ad hoc queries, but combining dynamic SQL with user input creates exposure that makes SQL-injection attacks possible. Replacing dynamic SQL with prepared SQL or stored procedures is feasible in most applications. Prepared SQL and stored procedures accept user input as parameter data rather than as SQL commands, thus limiting what an intruder can do. Of course, replacing dynamic SQL with a stored procedure won't help you if you use the user input to build dynamic SQL statements in your stored procedures. In that case, the dynamic SQL that the user input creates will still be corrupted, and your database will still be in danger of SQL-injection attack.

#### 1. Use Double Quotes

Replace all the single quotes that your users' input contains with double quotes. This simple precaution will go a long way toward warding off SQL-injection attacks. Single quotes often terminate SQL expressions and give the input more power than is necessary. Replacing the single quotes with double quotes will cause many SQL injection attacks to fail.

**QUESTION NO: 64**

**You are an application developer for Testkingprep.in. You are testing an application that was developed by another developer.**

**The application maintains its own list of authorized users. Each user is assigned a security level of 1, 2, or**

**3. When a new user account is created, the security level for that user is entered into a text box. The new user account information is saved in a Microsoft SQL Server table by using a stored procedure. You verify that user accounts that have any of the three security levels can perform only the intended actions within the application. You need to identify any security vulnerabilities in the portion of the application that creates new user accounts. What should you do?**

- A. Use SQL Query Analyzer to create a new user account that has a security level of 2. Test the application to see if the new user account can log on to the application.
- B. Create a new user account that has a security level other than 1, 2, or 3. Test the application to see what the new user account can do.
- C. Use Osq.exe to call the stored procedure and create a new user account that has a security level of 3. Test the application to see what the new user account can do.
- D. Create a new user account that has a security level of 3. Test the application to see what the new user account can do.

**Answer: B**

**Explanation**

Security testing is about validating your application's security services and identifying potential security flaws.

This section contains important testing recommendations for verifying that you have created a securable application.

Since attackers have no standard method of breaking into things, there are no standard methods of conducting

security testing. Also, there are few tools available at this time to test security aspects thoroughly. Since a

functional bug in an application can also represent a potential security flaw, you need to conduct functional

testing prior to conducting security testing.

It is important to note that security testing will not prove conclusively that an application is secure. Instead, it serves only to validate the effectiveness of instituted countermeasures, which were chosen based upon presumptions that were made during the threat analysis phase. Provided below are some suggestions for testing the securability of your application. There are some security issues you should be aware of when you test your smart documents. These security measures, described in the Security section, are in place to provide security for Microsoft® Office 2003 users. However, during testing, you may want to disable the XML expansion pack security check, if possible, or you may want to create a test environment that meets the security requirements of your users. The following topics provide additional information about security within a development and testing environment:

- Disabling the XML Expansion Pack Security Check
- Digital Code Signing for Testing Purposes
- Creating a Digital Certificate for Testing Purposes
- Delay Signing a Smart Document Assembly
- Testing a Signed XML Expansion Pack

### **Test for Buffer Overflows**

One of the first security bugs exploited in computer history was a buffer overflow. Buffer overflows continue to be one of the most dangerous and most commonly occurring weaknesses. Attempts to exploit this type of vulnerability can result in problems ranging from crashing the application to an attacker inserting and executing malignant code in the application process.

When writing data to buffers, it is imperative that developers not write more to the buffer than it can possibly hold. If the amount of data being written exceeds the buffer space that has been allocated, a buffer overflow occurs. When a buffer overflow occurs, data is written into parts of memory that may be allocated for other purposes. A worst-case scenario is when the buffer overflow contains malicious code that is then executed.

Buffer overflows account for a large percentage of security vulnerabilities.

### **Conduct source code security reviews**

Depending upon the sensitivity of the application in question, it might be prudent to conduct a security audit of the application source code. A source code audit should not be confused with a code review. The purpose of a standard code review is to identify general code defects that affect the functionality of the code. The purpose of

a source code security review is to identify security flaws, intentional or otherwise. Such a review would be especially warranted when developing applications that handle financial transactions or provide for public safety.

### **Validate contingency plans**

There will always be a potential that an application's security defenses can be breached and it is only prudent that contingency plans are in place and validated. What steps will be taken if a virus is detected on your application server or in your data center? When security is thwarted, reactions must occur rapidly to prevent further damage. Find out if your contingency plans will work before they must be battle-tested.

### **Attack your application**

Testers are accustomed to tormenting applications in an attempt to make them fail. Hacking your own application is a similar, but more focused, process. When attempting to attack your application, you should be looking for exploitable flaws that represent a weak spot in your application's defenses.