

NetPass: Membership and Role Management

Security Technology Is Only Part of the Solution

Implementing security is only part of the solution. Another important part is vigilance. Even if your system has many security safeguards, you need to watch it closely in these ways:

- Monitor your system's event logs. Watch for repeated attempts to log into your system or for excessive requests being made against your Web server.
- Continually keep your application server up to date with the latest security updates for Microsoft Windows and Internet Information Services (IIS), as well as any updates for Microsoft SQL Server or other data sources that your application might use.

Threat Modeling

An important part of developing a more secure application is to understand the threats to it. Microsoft has developed a way to categorize threats: Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege (STRIDE). The sections below briefly describe these threats and how they apply to Web applications.

Spoofing

To spoof is to impersonate a user or process in an unauthorized way. At its simplest, spoofing can mean typing in a different user's credentials. A malicious user might also change the contents of a cookie to pretend that he or she is a different user or that the cookie comes from a different server.

In general, you can help prevent spoofing by using stringent authentication. Any time someone requests access to non-public information, be sure they are who they say they are. You can also help defend against spoofing by keeping credential information safe. For example, do not keep a password or other sensitive information in a cookie, where a malicious user can easily find or modify it.

Tampering

Tampering means changing or deleting a resource without authorization. One example is defacing a Web page, where the malicious user gets into your site and changes files. An indirect way to tamper is by using a script exploit. A malicious user manages to get code (script) to execute by masking it as user input from a page or as a link.

A primary defense against tampering is to use Windows security to lock down files, directories, and other Windows resources. The application should also run with minimum privileges. You help guard against script exploits by not trusting any

information that comes from a user or even from a database. Whenever you get information from an untrusted source, take steps to be sure it does not contain any executable code.

Repudiation

A repudiation threat involves carrying out a transaction in such a way that there is no proof after the fact of the principals involved in the transaction. In a Web application, this can mean impersonating an innocent user's credentials. You can help guard against repudiation by using stringent authentication. In addition, use the logging features of Windows to keep an audit trail of any activity on the server.

Information Disclosure

Information disclosure simply means stealing or revealing information that is supposed to be private. A typical example is stealing passwords, but information disclosure can involve access to any file or resource on the server.

The best defense against information disclosure is to have no information to disclose. For example, if you avoid storing passwords, malicious users cannot steal them. An alternative to storing passwords is to store only a hash of the password. When a user presents credentials, you can hash the user's password and compare only the hashes of the two. If you do store sensitive information, use Windows security to help secure it. As always, you should use authentication to help ensure that only authorized users can access restricted information. If you must expose sensitive information, it is recommended that you encrypt the information when stored and use Secure Sockets Layer (SSL) to encrypt the information when sent to and from the browser.

Denial of Service

A denial of service attack is to deliberately cause an application to be less available than it should be. A typical example is to overload a Web application so that it cannot serve ordinary users. Alternatively, malicious users might try to simply crash your server.

IIS enables you to throttle applications, which means that it limits the number of requests it will serve. You might be able to deny access to users or IP addresses known to be malicious. Keeping your applications online is a matter of running robust code. You should test your application thoroughly and respond appropriately to error conditions wherever possible.

Elevation of Privilege

An elevation of privilege attack is to use malicious means to get more permissions than normally assigned. For example, in a successful elevation-of-privilege attack, a malicious user manages to get administrative privileges to your Web server, giving himself or herself access to any data on the server as well as control over server capabilities.

To help protect against elevation of privilege, run the application in a least-privilege context if practical. For example, it is recommended that you do not run ASP.NET applications as the SYSTEM (administrative) user.

Introduction to Membership

ASP.NET membership gives you a built-in way to validate and store user credentials. ASP.NET membership therefore helps you manage user authentication in your Web sites. You can use ASP.NET membership with ASP.NET Forms authentication or with the ASP.NET login controls to create a complete system for authenticating users.

ASP.NET membership supports facilities for:

- Creating new users and passwords.
- Storing membership information (user names, passwords, and supporting data) in Microsoft SQL Server, Active Directory, or an alternative data store.
- Authenticating users who visit your site. You can authenticate users programmatically, or you can use the ASP.NET login controls to create a complete authentication system that requires little or no code.
- Managing passwords, which includes creating, changing, and resetting them . Depending on membership options you choose, the membership system can also provide an automated password-reset system that takes a user-supplied question and response.
- Exposing a unique identification for authenticated users that you can use in your own applications and that also integrates with the ASP.NET personalization and role-management (authorization) systems.
- Specifying a custom membership provider, which allows you to substitute your own code to manage membership and maintain membership data in a custom data store

Membership, Roles and the User Profile

Although membership is a self-standing feature in ASP.NET for authentication, it can be integrated with ASP.NET role management to provide authorization services for your site. Membership can also be integrated with the user profile to provide application-specific customization that can be tailored to individual users.

How Membership Works

To use membership, you must first configure it for your site. In outline, you follow these steps:

1. Specify membership options as part of your Web site configuration. By default, membership is enabled. You can also specify what membership provider you

want to use. (In practical terms, this means that you are specifying what type of database you want to keep membership information in.) The default provider uses a Microsoft SQL Server database. You can also choose to use Active Directory to store membership information, or you can specify a custom provider (NetPass Membership Provider). For information on membership configuration options that can be specified in the Web.config file for your ASP.NET application, see *Configuring NetPass*.

2. Configure your application to use Forms authentication (as distinct from Windows or Passport authentication). You typically specify that some pages or folders in your application are protected and are accessible only to authenticated users.
3. Define user accounts for membership. You can do this in a variety of ways. You can use the Web Site Administration Tool, which provides a wizard-like interface for creating new users. Alternatively, you can create a "new user" ASP.NET Web page where you collect a user name and password (and optionally an e-mail address), and then use a membership function named CreateUser to create a new user in the membership system.

You can now use membership to authenticate users in your application. Most often, you will provide a login form, which might be a separate page or a special area on your home page. You can create the login form by hand using ASP.NET TextBox controls, or you can use ASP.NET login controls. Because you have configured the application to use Forms authentication, ASP.NET will automatically display the login page if an unauthenticated user requests a protected page.

Note: The ASP.NET login controls (Login, LoginView, LoginStatus, LoginName, and PasswordRecovery) encapsulate virtually all of the logic required to prompt users for credentials and validate the credentials in the membership system.

If you use login controls, they will automatically use the membership system to validate a user. If you have created a login form by hand, you can prompt the user for a user name and password and then call the ValidateUser method to perform the validation. After the user is validated, information about the user can be persisted (for example, with an encrypted cookie if the user's browser accepts cookies) using Forms Authentication. The login controls perform this task automatically. If you have created a login form by hand, you can call methods of the FormsAuthentication class to create the cookie and write it to the user's computer. If a user has forgotten his or her password, the login page can call membership functions that help the user remember the password or create a new one.

Each time the user requests another protected page, ASP.NET Forms authentication checks whether the user is authenticated and then either allows the user to view the page or redirects the user to the login page. By default, the authentication cookie remains valid for the user's session.

After a user has been authenticated, the membership system makes available an object that contains information about the current user. For example, you can get properties of the membership user object to determine the user's name and e-mail address, when the user last logged into your application, and so on.

An important aspect of the membership system is that you never need to explicitly perform any low-level database functions to get or set user information. For example, you create a new user by calling the membership `CreateUser` method. The membership system handles the details of creating the necessary database records to store the user information. When you call the `ValidateUser` method to check a user's credentials, the membership system does all the database lookup for you.

Membership Configuration and Management

You configure the membership system in your application's `Web.config` file. The easiest way to configure and manage membership is with the Web Site Administration Tool, which provides a wizard-based interface. As part of membership configuration, you specify:

- What membership provider to use. (This typically also specifies what database to store membership information in.)
- Password options such as encryption and whether to support password recovery based on a user-specific question.
- Users and passwords. If you are using the Web Site Administration Tool, you can create and manage users directly. Otherwise, you must call membership functions to create and manage users programmatically.

Configuring NetPass

ASP.NET Membership is configured using the membership element in the `Web.config` file for your application. The membership element is a sub-element of the `system.web` section. You can enable ASP.NET Membership for an application by directly editing the `Web.config` file for that application, or you can use the Web Site Administration Tool, which provides a wizard-based interface. As part of membership configuration, you specify:

- Which membership provider (or providers) to use. (This typically also specifies what database to store membership information in.)
- Password options such as encryption and whether to support password recovery based on a user-specific question.
- Users and passwords. If you are using the Web Site Administration Tool, you can create and manage users directly. Otherwise, you must call membership functions to create and manage users programmatically.

```
<connectionStrings>
  <add name="NetPassServices"
        connectionString="DRIVER={SQL Native Client}; ... />
</connectionStrings>

<roleManager defaultProvider="NetPassRoleProvider"
              enabled="true"
              cacheRolesInCookie="true">
```

```
        cookieName=".ASPROLES"
        cookieTimeout="30"
        cookiePath="/"
        cookieRequireSSL="false"
        cookieSlidingExpiration="true"
        cookieProtection="All" >
    <providers>
        <clear />
        <add
            name="NetPassRoleProvider"
            type="WDK.CommunityServices.NetPass.NetPassRoleProvider"
            connectionString="NetPassServices"
            applicationName="NetPass"
            writeExceptionsToEventLog="false" />
    </providers>
</roleManager>

<membership defaultProvider="NetPassMembershipProvider"
userIsOnlineTimeWindow="15">
    <providers>
        <clear />
        <add
            applicationName="NetPass"
            name="NetPassMembershipProvider"
            type="WDK.CommunityServices.NetPass.NetPassMembershipProvider"
            connectionString="NetPassServices"
            passwordFormat="Clear"
            enablePasswordRetrieval="true"
            enablePasswordReset="true"
            requiresQuestionAndAnswer="true"
            writeExceptionsToEventLog="true" />
    </providers>
</membership>
```