.NET Framework 4 - ASP.NET

**Basic Security Practices for Web Applications**

The topic of creating a secure Web application is extensive. It requires study to understand security vulnerabilities. You also need to familiarize yourself with the security facilities of Windows, the .NET Framework, and ASP.NET. Finally, it is essential to understand how to use these security features to counter threats.

Even if you are not experienced with security, there are basic measures that you should take to protect your Web application. The following list provides minimum-security guidelines that apply to all Web applications and that you should follow:

* [General Web Application Security Recommendations](http://msdn.microsoft.com/en-us/library/zdh19h94.aspx#cpconbestsecuritypracticesforwebapplicationsanchor1)
* [Run Applications with Least Privileges](http://msdn.microsoft.com/en-us/library/zdh19h94.aspx#cpconbestsecuritypracticesforwebapplicationsanchor2)
* [Know Your Users](http://msdn.microsoft.com/en-us/library/zdh19h94.aspx#cpconbestsecuritypracticesforwebapplicationsanchor3)
* [Guard Against Malicious User Input](http://msdn.microsoft.com/en-us/library/zdh19h94.aspx#cpconbestsecuritypracticesforwebapplicationsanchor4)
* [Access Databases Securely](http://msdn.microsoft.com/en-us/library/zdh19h94.aspx#cpconbestsecuritypracticesforwebapplicationsanchor5)
* [Create Safe Error Messages](http://msdn.microsoft.com/en-us/library/zdh19h94.aspx#cpconbestsecuritypracticesforwebapplicationsanchor6)
* [Keep Sensitive Information Safely](http://msdn.microsoft.com/en-us/library/zdh19h94.aspx#cpconbestsecuritypracticesforwebapplicationsanchor7)
* [Use Cookies Securely](http://msdn.microsoft.com/en-us/library/zdh19h94.aspx#cpconbestsecuritypracticesforwebapplicationsanchor8)
* [Guard Against Denial-of-Service Threats](http://msdn.microsoft.com/en-us/library/zdh19h94.aspx#cpconbestsecuritypracticesforwebapplicationsanchor9)

|  |
| --- |
| **Description: NoteNote** |
| For comprehensive and detailed security guidance which will help you to design, develop, configure, and deploy more secure ASP.NET Web applications, see the security modules provided on the [Microsoft Patterns and Practices](http://go.microsoft.com/fwlink/?LinkId=37129) Web site. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifGeneral Web Application Security Recommendations

Even the most elaborate application security can fail if a malicious user can use simple ways to get to your computers. Follow these guidelines:

* Back up often and keep your backups physically secure.
* Keep your Web server computer physically secure so that unauthorized users cannot get to it, turn it off, or take it.
* Use the Windows NTFS file system, not FAT32. NTFS offers substantially more security than FAT32. For details, see the Windows documentation.
* Secure the Web server computer and all computers on the same network with strong passwords.
* Secure IIS. For details, see the [Microsoft TechNet Security Center](http://go.microsoft.com/fwlink/?LinkId=50530) Web site.
* Close unused ports and turn off unused services.
* Run a virus checker that monitors inbound and outbound traffic.
* Establish and enforce a policy that forbids users from keeping their passwords written down in an easy-to-find location.
* Use a firewall. For recommendations, see [Microsoft Firewall Guidelines](http://go.microsoft.com/fwlink/?LinkId=50217) on the Microsoft security site.
* Install the latest security patches from Microsoft and other vendors. For example, the see the [Microsoft TechNet Security Center](http://go.microsoft.com/fwlink/?LinkId=50530) Web site has a list of the latest security bulletins for all Microsoft products. Other vendors have similar sites.
* Use Windows event logging and examine the logs frequently for suspicious activity. This includes repeated attempts to log on to your system and an extremely high number of requests against your Web server.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifRun Applications with Least Privileges

When your application runs, it runs within a context that has specific privileges on the local computer and potentially on remote computers. For information on configuring the application identity, see [Configuring ASP.NET Process Identity](http://msdn.microsoft.com/en-us/library/dwc1xthy.aspx). To run with least privileges, follow these guidelines:

* Do not run your application with the identity of a system user (administrator).
* Run the application in the context of a user with the minimum practical privileges.
* Set permissions (Access Control Lists or ACLs) on all the resources required for your application. Use the least permissive setting. For example, if practical in your application, set files to be read-only. For a list of the minimum required ACL permissions required for the identity of your ASP.NET application, see [ASP.NET Required Access Control Lists (ACLs)](http://msdn.microsoft.com/en-us/library/kwzs111e.aspx).
* Keep files for your Web application in a folder below the application root. Do not allow users the option of specifying a path for any file access in your application. This helps prevent users from getting access to the root of your server.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifKnow Your Users

In many applications, users access the site anonymously (without having to provide credentials). If so, your application accesses resources by running in the context of a predefined user. By default, this context is the local ASPNET user (on Windows 2000 or Windows XP) or NETWORK SERVICE user (on Windows Server 2003) on the Web server computer. To restrict access to users who are authenticated, follow these guidelines:

* If your application is an intranet application, configure it to use Windows integrated security. That way, the user's logon credentials can be used to access resources. For more information, see [ASP.NET Impersonation](http://msdn.microsoft.com/en-us/library/xh507fc5.aspx).
* If you need to gather credentials from the user, use one of the ASP.NET authentication strategies. For an example, see [Managing Users by Using Membership](http://msdn.microsoft.com/en-us/library/tw292whz.aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifGuard Against Malicious User Input

As a general rule, never assume that input you get from users is safe. It is easy for malicious users to send potentially dangerous information from the client to your application. To guard against malicious input, follow these guidelines:

* In ASP.NET Web pages, filter user input to check for HTML tags, which might contain script. For details, see [How to: Protect Against Script Exploits in a Web Application by Applying HTML Encoding to Strings](http://msdn.microsoft.com/en-us/library/a2a4yykt.aspx).
* Never echo (display) unfiltered user input. Before displaying untrusted information, encode HTML to turn potentially harmful script into display strings.
* Never store unfiltered user input in a database.
* If you want to accept some HTML from a user, filter it manually. In your filter, explicitly define what you will accept. Do not create a filter that tries to filter out malicious input; it is very difficult to anticipate all possible malicious input.
* Do not assume that information you get from the HTTP request header (in the [HttpRequest](http://msdn.microsoft.com/en-us/library/system.web.httprequest.aspx) object) is safe. Use safeguards for query strings, cookies, and so on. Be aware that information the browser reports to the server (user agent information) can be spoofed, in case that is important in your application.
* If possible, do not store sensitive information in a place accessible from the browser, such as hidden fields or cookies. For example, do not store a password in a cookie.

|  |
| --- |
| **Description: NoteNote** |
| View state is stored in a hidden field in an encoded format. By default, it includes a message authentication code (MAC) so that the page can determine whether view state was tampered with. If sensitive information is stored in view state, encrypt by setting the page's [ViewStateEncryptionMode](http://msdn.microsoft.com/en-us/library/system.web.ui.page.viewstateencryptionmode.aspx) property to **true**. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifAccess Databases Securely

Databases typically have their own security. An important aspect of a secure Web application is designing a way for the application to access the database securely. Follow these guidelines:

* Use the inherent security of your database to limit who can access database resources. The exact strategy depends on your database and your application:
  + If practical in your application, use integrated security so that only Windows-authenticated users can access the database. Integrated security is more secure than passing explicit credentials to the database.
  + If your application involves anonymous access, create a single user with very limited permissions, and perform queries by connecting as this user.
* Do not create SQL statements by concatenating strings that involve user input. Instead, create a parameterized query and use user input to set parameter values.
* If you must store a user name and password somewhere to use as the database login credentials, store them in the Web.config file and secure the file with protected configuration. For details, see [Encrypting Configuration Information Using Protected Configuration](http://msdn.microsoft.com/en-us/library/53tyfkaw.aspx).

For more information on accessing data securely, see [Securing Data Access](http://msdn.microsoft.com/en-us/library/ms178375.aspx) and [Securing ADO.NET Applications](http://msdn.microsoft.com/en-us/library/ecb3hak0.aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifCreate Safe Error Messages

If you are not careful, a malicious user can deduce important information about your application from the error messages it displays. Follow these guidelines:

* Do not write error messages that echo information that might be useful to malicious users, such as a user name.
* Configure the application not to show detailed errors to users. If you want to display detailed error messages for debugging, determine first whether the user is local to the Web server. For details, see [How to: Display Safe Error Messages](http://msdn.microsoft.com/en-us/library/994a1482.aspx).
* Use the [customErrors](http://msdn.microsoft.com/en-us/library/h0hfz6fc.aspx)configuration element to control who can view exceptions from the server.
* Create custom error handling for situations that are prone to error, such as database access. For more information, see [Error Handling in ASP.NET Pages and Applications](http://msdn.microsoft.com/en-us/library/w16865z6.aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifKeep Sensitive Information Safely

Sensitive information is any information that you need to keep private. A typical piece of sensitive information is a password or an encryption key. If a malicious user can get to the sensitive information, then the data protected by the secret is compromised. Follow these guidelines:

* If your application transmits sensitive information between the browser and the server, consider using the Secure Sockets Layer (SSL). For details on how to secure a site with SSL, see article Q307267, "HOW TO: Secure XML Web Services with Secure Socket Layer in Windows 2000" in the Microsoft Knowledge Base at http://support.microsoft.com.
* Use protected configuration to secure sensitive information in configuration files such as the Web.config or Machine.config files. For more information, see [Encrypting Configuration Information Using Protected Configuration](http://msdn.microsoft.com/en-us/library/53tyfkaw.aspx).
* If you must store sensitive information, do not keep it in a Web page, even in a form that you think people will not be able to see it (such as in server code).
* Use the strong encryption algorithms supplied in the [System.Security.Cryptography](http://msdn.microsoft.com/en-us/library/system.security.cryptography.aspx) namespace.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifUse Cookies Securely

Cookies are a useful way to keep user-specific information available. However, because cookies are sent to the browser's computer, they are vulnerable to spoofing or other malicious use. Follow these guidelines:

* Do not store any critical information in cookies. For example, do not store a user's password in a cookie, even temporarily. As a rule, do not keep anything in a cookie that, if spoofed, can compromise your application. Instead, keep a reference in the cookie to a location on the server where the information is.
* Set expiration dates on cookies to the shortest practical time you can. Avoid permanent cookies if possible.
* Consider encrypting information in cookies.
* Consider setting the [Secure](http://msdn.microsoft.com/en-us/library/system.web.httpcookie.secure.aspx) and [HttpOnly](http://msdn.microsoft.com/en-us/library/system.web.httpcookie.httponly.aspx) properties on the cookie to **true**.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifGuard Against Denial-of-Service Threats

An indirect way that a malicious user can compromise your application is by making it unavailable. The malicious user can keep the application too busy to service other users, or if can simply cause the application to crash. Follow these guidelines:

* Use error handling (for example, try-catch). Include a finally block in which you release resources in case of failure.
* Configure IIS to use process throttling, which prevents an application from using up a disproportionate amount of CPU time.
* Test size limits of user input before using or storing it.
* Put size safeguards on database queries. For example, before you display query results in an ASP.NET Web page, be sure that there are not an unreasonable number of records.
* Put a size limit on file uploads, if those are part of your application. You can set a limit in the Web.config file using syntax such as the following, where the maxRequestLength value is in kilobytes:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl46_ctl00_ctl00_code');" \o "Copy Code)

<configuration>

<system.web>

<httpRuntime maxRequestLength="4096" />

</system.web>

</configuration>

You can also use [RequestLengthDiskThreshold](http://msdn.microsoft.com/en-us/library/system.web.configuration.httpruntimesection.requestlengthdiskthreshold.aspx) property in to reduce the memory overhead of large uploads and form posts.

.NET Framework 4 - ASP.NET

**Overview of Web Application Security Threats**

If unknown users can access your Web application, the odds are almost certain that malicious users will try to gain unauthorized access to your application. Servers that are accessible to the public on the Internet are constantly probed for vulnerabilities. Therefore, it is recommended that you take precautions and build security into all of your Web applications.

More detailed information about best practices for writing secure code and securing applications can be found in the book "Writing Secure Code" by Michael Howard and David LeBlanc, or through the guidance provided by [Microsoft Patterns and Practices](http://go.microsoft.com/fwlink/?linkid=37129).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifSecurity 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.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifThreat 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 uses 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.