**Best practices for input validation in ASP.NET Core:**

* Validate input

Always validate user input before sending it across the application.

* Client-side and server-side validation

Perform input validation on both the client and server side. Client-side validation can help reduce server load, but it's not a substitute for server-side validation.

* Fluent validation

Use Fluent Validation to enforce complex business rules, such as validating that a reservation date is not in the past.

* Implement error handling

Implement custom error handling to prevent sensitive information from leaking and to provide a user-friendly response to unexpected situations.

* Secure session management

Use session tokens with secure properties, such as randomness and expiration. Also, implement session timeouts and enforce re-authentication for sensitive actions.

* File upload validation

Validate file extensions to prevent attackers from uploading malicious script files.

* Employ security testing protocols

Integrate security testing protocols into your development workflow to identify and fix vulnerabilities.

**ASP.NET security best practices**

Curious about the key strategies to ensure the security and reliability of your ASP.NET applications, including [building APIs](https://dotnet.microsoft.com/en-us/apps/aspnet/apis?ref=escape.tech)? Dive into our latest blog post, where we guide you through ASP.NET security best practices. Explore how these practices can not only enhance the security of your web applications but also bring tangible benefits to your development journey.

In this guide, [Escape](https://escape.tech/?ref=escape.tech)'s security research team has gathered 10 crucial ASP.NET security best practices that empower you to create more resilient and efficient ASP.NET projects. Let's get started!

1. [**What is ASP.NET?**](https://escape.tech/blog/asp-dot-net-security/#what-is-aspnet)
2. [ASP.NET security best practices: 1. Enforce HTTPS](https://escape.tech/blog/asp-dot-net-security/#aspnet-security-best-practices-1-enforce-https)
3. [2. Use Anti-Forgery Tokens](https://escape.tech/blog/asp-dot-net-security/#2-use-anti-forgery-tokens)
4. [3. Validate input](https://escape.tech/blog/asp-dot-net-security/#3-validate-input)
5. [4. Implement Authentication and Authorization](https://escape.tech/blog/asp-dot-net-security/#4-implement-authentication-and-authorization)
6. [5. Secure Session Management](https://escape.tech/blog/asp-dot-net-security/#5-secure-session-management)
7. [6. Encrypt configuration files](https://escape.tech/blog/asp-dot-net-security/#6-encrypt-configuration-files)
8. [7. Implement error handling](https://escape.tech/blog/asp-dot-net-security/#7-implement-error-handling)
9. [8. Regularly update framework and libraries](https://escape.tech/blog/asp-dot-net-security/#8-regularly-update-framework-and-libraries)
10. [9. Logging and monitoring](https://escape.tech/blog/asp-dot-net-security/#9-logging-and-monitoring)
11. [10. Perform proper security testing](https://escape.tech/blog/asp-dot-net-security/#10-perform-proper-security-testing)
12. [Conclusion](https://escape.tech/blog/asp-dot-net-security/#conclusion)
13. [FAQ](https://escape.tech/blog/asp-dot-net-security/#faq)

**What is ASP.NET?**

ASP.Net core is a popular framework for creating web applications and APIs using Microsoft's .NET runtime.

Due to their usage in many enterprise companies, ASP.NET applications often handle sensitive data or have strong requirements for availability. In this context, best practices for security are not just a feature; they are a necessity.

**ASP.NET security best practices:  
1. Enforce HTTPS**

**Why it's important**

Using HTTPS ensures that the data between your server and the client is encrypted, making it more difficult for attackers to intercept or tamper with data.

**How to implement**

In your ASP.NET application, configure the RequireHttpsAttribute in your Global.asax file:

protected void Application\_Start()

{

GlobalFilters.Filters.Add(new RequireHttpsAttribute());

}

**2. Use Anti-Forgery Tokens**

**Why it's important**

[CSRF attacks exploit](https://escape.tech/blog/understanding-and-dealing-with-cross-site-request-forgery-attacks/) the trust that a web application has in an authenticated user's browser, allowing attackers to perform unauthorized actions on behalf of the user. For instance, if a user is authenticated in one tab of their browser and visits a malicious site in another, the malicious site can send requests to the authenticated site without the user's knowledge. These requests can change user settings, post content, or even initiate transactions.

Anti-forgery tokens prevent Cross-Site Request Forgery (CSRF) attacks by ensuring that the requests sent to your server are from legitimate users.

**How to implement**

In your Razor views, use the Html.AntiForgeryToken() method:

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<!-- Form content goes here -->

}

**3. Validate input**

**Why it's important**

Input validation is a cornerstone of web application security, particularly in ASP.NET applications.

The importance of this practice comes from its role in defending against various forms of injection attacks, such as [*SQL injection*](https://escape.tech/blog/sql-injection-in-graphql/), *Cross-Site Scripting (XSS)*, and others. When user input is not properly validated or sanitized, attackers can exploit these vulnerabilities to inject malicious code into your system.

💡

[CSRF vs XSS: What is the difference?](https://escape.tech/blog/csrf-vs-xss/)

Consequences from the exploitation of injection vulnerabilities can be as disastrous as having your whole database erased by attackers. This is why, at Escape, we also created a [specific set of security tests](https://docs.escape.tech/injection?ref=escape.tech) to help you find and fix injections in your applications.

**How to implement**

Use ASP.NET's built-in validation controls or data annotations:

[Required]

[Display(Name = "Username")]

public string UserName { get; set; }

**4. Implement Authentication and Authorization**

**Why it's important**

Strong authentication and authorization ensure that only legitimate users can access certain parts of your application.

Authentication issues are so well spread among web applications that they have become a part of [OWASP's Top 10 vulnerabilities](https://escape.tech/blog/owasp-api-security-checklist-for-2023/) in web applications.

**How to implement**

You can use ASP.NET Identity for implementing role-based security in your ASP.NET applications:

[Authorize(Roles = "Admin")]

public ActionResult AdminOnly()

{

return View();

}

**5. Secure Session Management**

**Why it's important**

Secure session management is vital to protect user data and prevent unauthorized access.

Each of your user sessions contains sensitive information, such as user preferences, authentication tokens, and other personal data.

If this session data is not properly secured, it can be vulnerable to attacks like session hijacking or fixation, where attackers gain control of a user's session and thereby access their account and associated privileges. This can lead to data breaches, identity theft, and unauthorized actions performed on behalf of the user.

Implementing secure session management practices, such as using secure, HTTP-only cookies and storing session data on the server side, significantly reduces these risks. It ensures that session tokens are not easily intercepted or tampered with, maintaining the integrity and confidentiality of user sessions.

**How to implement**

Store sensitive information in server-side sessions and use secure, HTTP-only cookies.

Session["UserId"] = user.Id;

**6. Encrypt configuration files**

**Why it's important**

Encrypting configuration files in ASP.NET is a critical security measure.

Configuration files often contain sensitive data, such as database connection strings and API keys, which are essential for the application's operations.

If these files are left unencrypted, they become an easy target for attackers who gain access to the file system. Once compromised, this sensitive information can be used to launch further attacks, access databases, and misuse API services.

Encrypting these files adds a robust layer of security, making it much harder for unauthorized individuals to decipher the contents even if they gain access to the files. This practice is a straightforward yet effective way to safeguard important credentials and configuration details, ensuring they remain confidential and secure.

**How to implement**

Use the aspnet\_regiis.exe tool to encrypt sections of your Web.config file.

For example, to encrypt the connectionStrings section of an application located at /MyApp, use:

aspnet\_regiis -pe "connectionStrings" -app "/MyApp"

**7. Implement error handling**

**Why it's important**

When an application encounters an error, the default behavior might be to display a detailed error message. These messages can include sensitive information such as stack traces, database queries, or internal system details, which could be invaluable to a potential attacker.

By exposing such information, the application becomes vulnerable to various attacks, including SQL injection, system exploitation, and more.

Detailed error messages can also lead to poor user experience, causing confusion and mistrust. Implementing custom error handling not only prevents the leakage of sensitive information but also allows for a more controlled and user-friendly response to unexpected situations.

This approach ensures that users see a standardized error page that doesn't give away any internal workings of the application, thus maintaining both security and professionalism.

**How to implement**

Use custom error pages and avoid detailed error messages in production:

<customErrors mode="On" defaultRedirect="ErrorPage.html"/>

**8. Regularly update framework and libraries**

**Why it's important**

Regularly updating the framework and libraries in ASP.NET applications is a key security practice.

This is because updates often include patches for known vulnerabilities.

As security threats evolve and new vulnerabilities are discovered, the developers of these frameworks and libraries release updates to address these issues. By not updating, you leave your application exposed to known vulnerabilities, which attackers can easily exploit.

**How to implement**

Regularly check for and install updates to ASP.NET and third-party libraries. [OWASP Dependency check](https://owasp.org/www-project-dependency-check/?ref=escape.tech) can help you find and update outdated dependencies in your ASP.NET projects.

**9. Logging and monitoring**

**Why it's important**

Logging and monitoring are essential components in the security infrastructure of any web application.

They are important for your security team to detect and respond to security incidents.

By maintaining comprehensive logs, you create a detailed record of all activities and events occurring within your application. This information is invaluable when it comes to understanding the context and scope of any security breach. In the event of an attack or a security anomaly, these logs can be analyzed to pinpoint how the incident occurred, what vulnerabilities were exploited, and what parts of the system were affected.

Even more important, proactive monitoring of these logs enables you to identify and respond to potential security threats in real-time. It acts as an early warning system, alerting you to unusual or suspicious activities that could indicate a security breach or an attempted attack.

**How to implement**

Implement logging using libraries like [NLog](https://nlog-project.org/?ref=escape.tech) or [Serilog](https://serilog.net/?ref=escape.tech) and regularly monitor logs for suspicious activities.

**10. Perform proper security testing**

**Why it's important**

Most critical security bugs can only be found through proper security testing and assessment.

💡

Our security research team at Escape found that [more than 90% of public APIs](https://escape.tech/resources/state-of-public-apis-2023/?ref=escape.tech) had one or more security vulnerabilities.

Adding security tests as part of your regular software testing process is **critical.**

**How to implement**

You can use .NET testing frameworks like [Nunit](https://nunit.org/?ref=escape.tech) to write your custom security tests.

using NUnit.Framework;

using Moq;

using System.Web.Mvc;

using MyApplication.Controllers;

using System.Security.Principal;

using System.Web;

using System.Web.Routing;

namespace MyApplication.Tests

{

[TestFixture]

public class SecureControllerTests

{

[Test]

public void TestAdminOnlyAction\_AccessedByAdmin\_ReturnsView()

{

// Write test and assertions here

}

}

}

but this approach can be time-consuming, especially since you need to create and maintain all the test cases for many types of security bugs.

**Conclusion**

By following these guidelines, developers can significantly enhance the security posture of their applications, ensuring a safer experience for all users.

A faster and a more reliable alternative is to use a dedicated [API security testing tool with ASP.NET support like Escape](https://escape.tech/?ref=escape.tech) which streamlines this process by quickly identifying vulnerabilities, and providing actionable remediation, all without the need for traffic monitoring. With a low time to value of just 15 minutes, you can start enhancing your ASP.NET security almost instantly.