



*This is a living document. Download the most recent version at [www.sans-ssi.org](http://www.sans-ssi.org). Please send suggestions and comments to [spa@sans.org](mailto:spa@sans.org)*

## **GSSP (GIAC Secure Software Programmer)**

.NET Implementation Issues, Version 2

[www.sans.org](http://www.sans.org)

July 2008

**Task 1 – Data Handling** .NET programmers must be able to write programs that handles input, properly validates, and output the data. Programmers should be familiar with these attack scenarios: Cross-site Scripting (XSS) and SQL-Injection.

**01.1.1: Input Validation Principles.** .NET programmers must understand that input validation is an important part of building a trust boundary. Consequently input should not be trusted, regardless of its source. Programmers must also understand the white-list and black-list approaches and the tradeoffs between them. Programmers also need to know when and where to validate, not just what to validate. Programmers must understand strong typing vs. weak typing vs. no typing and what that means for Input Validation threats. .NET programmers must also understand threats with canonicalization.

**01.1.2: Input Validation Sources.** .NET programmers must recognize common sources of input to .NET applications. For example, HTTP requests, serialized streams, configuration files, backend databases, etc.

**01.1.3: Input Validation Techniques.** .NET programmers must understand how to validate both simple and complex data types. .NET programmer must understand the benefits of, and the effective use of the ASP.Net validation controls including ASP.NET page-based validation and validation techniques with ASP.NET AJAX. .NET programmers must also understand the Windows Forms event driven validation, error provider, and System.Windows.Forms.MaskedTextBox control. Familiarity with regular expression and the regular expression facilities provided by the .NET framework. .NET programmers should understand the advantages and disadvantages of using DTDs or XML schema validation for XML data.

**01.1.4: Output Encoding.** .NET programmers must understand when and how to use output encoding for both XML and HTML data. Dot.NET programmers must understand how to use Microsoft Anti-XSS library and understand the encoding facilities provided by the standard ASP.NET controls.

**01.1.5: Database Access.** .NET programmers must understand the security risks introduced by using dynamic queries with un-trusted data. Programmers must understand advantages and disadvantages of using parameterized queries, stored procedures, extended stored procedures and Microsoft's LINQ technology.

**Task 2 – Authentication and Authorization.** .NET applications often require making security decisions based on an identity. End-users or external programs must be able to prove their identity. .NET applications often require restricting access to assets and functionality based on pre-defined security policies. Active enforcement of these Authorization rules must be ensured by the programmer. An understanding of Authentication and Authorization in different tiers of an application is necessary. This

Task focuses on application-level Authentication and Authorization activities. Programmers should be familiar with the following attack scenarios: Phishing, eavesdropping, Man-In-The-Middle, and the passerby attack (shoulder-surfing).

**01.2.1: Authenticate Principles.** Programmers must understand authentication principles. For example when to authenticate (trust-boundary), C.A.P.T.C.H.A., multi-factor authentication, and re-authentication.

**01.2.2: Authentication Protection.** .NET programmers are required to know how to use encryption and certificates to protect various authentication processes. This includes an understanding of credential strength-of-function, credential expiration, account lockout and credential recovery/reset. .NET programmers must also understand how to use HTTP and HTML properties to protect Web-based authentication. For example, the Pragma header, the cache controls, and the password input type.

**01.2.3: Authentication Techniques.** .NET programmers must be familiar with the more common authentication techniques and APIs available within .NET Framework. This includes backend authentication techniques and principles, and various front-end authentication techniques such as Forms, Windows and Passport authentication. This familiarity assumes the programmer will understand the threats and tradeoffs for each technique. .NET programmers must understand the security implications of using ASP.NET Forms authentication including the use of enableCrossAppRedirects in version 2.0 and higher as well as lack of this option in version 1.0 and 1.1.

**01.2.4: What to Protect.** Resources and Functions. Programmers must be able to actively protect accesses to system data objects (resources) and system functionality (functions). An example resource is a user data object that contains information private to a specific user. An example function is a URI or other calls to privileged functions; user administration, business transactions, etc.

**01.2.5: How to Protect.** Declarative vs. Imperative. Programmers should understand access schemes that are defined via configuration files or attributes (Declarative) and schemes that are defined by active checks in custom code (Imperative).

**01.2.6: Access Control APIs.** .NET developers must understand the ASP.NET Membership and Role Providers. .NET developers must understand the difference between Principals and Identities.

**Task 3 –State Management.** .NET programmers must understand the principles of state management and the differences between client side and server side state management.

**01.3.1: Session Protection.** For the protection of session tokens, .NET programmers are required to understand the implications of several topics including encryption, token strength-of-function, active and inactive timeouts, and re-issuance. Programmers should be familiar with cookie management and cookie protection on the client and server including persistent vs. session cookies, and the following attributes: HTTPOnly, Secure, Domain, and Path. Programmers should be familiar with the following attack scenarios: Cross-Site Request Forgery (aka One-Click Attack), Session Fixation, and various methods of Session Hijacking.

**01.3.2: Session State Management.** .NET programmers must understand the differences between the different Session State modes and the advantages and disadvantages of each.

**01.3.3: View State.** .NET application developers must understand how to utilize .NET's view state and how this affects the security of state data. .NET developers must understand the different methods of protection available for the view state and the differences between the default configurations of .NET 1.0 vs. later versions.

**Task 4 – .NET Types.** .NET programmers must understand the security implications of built-in data types and .NET-specific memory management.

**01.4.1: String Immutability.** .NET programmers must understand the immutability property of System.String and how to effectively use the System.Security.SecureString class.

**01.4.2: Integer and Double Overflows.** .NET programmers must understand the limitations of .NET's numerical data types and the resulting security implications.

**01.4.3: String Comparisons.** .NET programmers must understand how to properly compare two System.String objects in a language and locale independent way.

**01.4.4: Class-level Security.** .NET programmers must be familiar with accessibility modifiers (*public*, *private*, *protected*, *internal*, *sealed*, *unsafe*) and how they can be used to protect classes, class members, and class methods. Programmers should also understand class comparisons, *serialization*, *clone*-ability, and nested classes.

**Task 5 – Application Faults & Logging.** All .NET application programmers need to be able to properly handle expected and unexpected application faults and understand secure logging principles.

**01.5.1: Exception Handling.** .NET application developers must understand try/catch/finally construct to appropriately handle exceptions to prevent resource leaks and other undesired behaviors. Developers must also understand the concepts of fail open and fail closed when handling exceptions.

**01.5.2: Logging.** Developers must understand the principles behind logging security-relevant events such as login, logoff, credential changes, exceptions, etc. Developers must understand what information should and should not be logged in order to ensure non-repudiation, reconstruct an attack, or prevent sensitive information from being logged. .NET Developers should understand logging frameworks including ASP.NET Health Monitoring.

**01.5.3: Configuration of Error Handling.** .NET developers should be familiar with the configuration and use of customErrors and errorPage.

**Task 6 – Encryption Services.** .NET programmers must understand when and how to use encryption to protect sensitive data and manage keys used to maintain confidentiality, integrity, and non-repudiation.

**01.6.1: Communications Encryption.** .NET Developers are responsible for knowing which of an application's external connections should be protected with encryption and how to utilize the cryptographic services provided by the .NET Framework to protect the communication channel.

**01.6.2: Encryption of Data at Rest.** .NET developers must understand how to store sensitive data in an encrypted format. This includes familiarity with common encryption API's, .NET random number generation, and the facilities within the framework for encrypting configuration data.

**Task 7 – Secure Architecture & Coding Principles.** Programmers must understand architecture-level issues and coding practices that contribute to security.

**01.7.1: Thread Safety.** .NET application developers must understand race conditions, deadlock, starvation, and how each can affect system security. .NET programmers must also be able to effectively use the collection classes in a thread safe manner. Programmers must be aware of the multi-threading considerations for static class members, instance variables, and volatile fields.

**01.7.2: Singletons.** .NET developers must understand when a Singleton is needed and how to implement the Singleton pattern in .NET.

**01.7.3: .NET HTTP Modules and HTTP Handlers.** .NET programmers must be familiar with HTTP Modules and HTTP Handlers and how they can be used to increase the security of ASP.NET applications.

**01.7.4: Secure Coding Principles.** Programmers should be familiar with technology independent security coding principles. For example: least privilege, secure initialization, separation of data from code (no hard-coding values into programs), type-checking, shallow vs. deep copy, global variables, securely formed *if* and *while* statements (always using comparators and putting constant values first in the comparison), bounds checking, function complexity, off-by-one errors, encapsulation, and the differences between managed and unmanaged code.

**01.7.5: Object Serialization:** .NET developers must understand the security impact of serialization and how to safely and securely enable this functionality.

**Task 8 – Code Access Security.** .NET Programmers must understand how to effectively use Code Access Security.

**01.8.1: Code Privileges.** .NET Programmers must understand how to manage the privileges of code as well as the different protection domains. This includes an understanding of the run-time security policies, trust levels, evidence, strong names, authenticode, Global Assembly Cache (GAC), partially trusted callers, assembly requests.

**01.8.2: Declarative and Imperative Permissions:** .NET application developers must understand how to use declarative and imperative permissions to limit the permissions that code requires at the assembly, class, and method levels.

==end==