UNCLASSIFIED

MICROSOFT (MS) .NET FRAMEWORK 4.0 SECURITY TECHNICAL IMPLEMENTATION GUIDE (STIG) OVERVIEW

Version 2, Release 1

22 January 2021

Developed by DISA for the DoD

UNCLASSIFIED

Trademark Information

TABLE OF CONTENTS

LIST OF TABLES

1. INTRODUCTION

1.1 Executive Summary

The Microsoft .NET Framework 4.0 Security Technical Implementation Guide (STIG) provides guidance for secure configuration and usage of Microsoft's .NET Framework version 4.0. The STIG provides security guidance for .NET deployments in workstations or servers and focuses on the secure configuration of the .NET Common Language Runtime (CLR). This overview document gives technology-specific background and information on conducting a security review for .NET Framework Version 4.0. Previous versions of .NET are not addressed specifically, although some of the information may significantly overlap with previous versions.

1.2 Authority

DoD Instruction (DoDI) 8500.01 requires that "all IT that receives, processes, stores, displays, or transmits DoD information will be […] configured […] consistent with applicable DoD cybersecurity policies, standards, and architectures" and tasks that Defense Information Systems Agency (DISA) "develops and maintains control correlation identifiers (CCIs), security requirements guides (SRGs), security technical implementation guides (STIGs), and mobile code risk categories and usage guides that implement and are consistent with DoD cybersecurity policies, standards, architectures, security controls, and validation procedures, with the support of the NSA/CSS, using input from stakeholders, and using automation whenever possible." This document is provided under the authority of DoDI 8500.01.

Although the use of the principles and guidelines in these SRGs/STIGs provide an environment that contributes to the security requirements of DoD systems, applicable NIST SP 800-53 cybersecurity controls need to be applied to all systems and architectures based on the Committee on National Security Systems (CNSS) Instruction (CNSSI) 1253.

1.3 Vulnerability Severity Category Code Definitions

Severity Category Codes (referred to as CAT) are a measure of vulnerabilities used to assess a facility or system security posture. Each security policy specified in this document is assigned a Severity Category Code of CAT I, II, or III.

Table 1-1: Vulnerability Severity Category Code Definitions

	DISA Category Code Guidelines
CAT I	Any vulnerability, the exploitation of which will, **directly and immediately** result in loss of Confidentiality, Availability, or Integrity.
CAT II	Any vulnerability, the exploitation of which **has a potential** to result in loss of Confidentiality, Availability, or Integrity.
CAT III	Any vulnerability, the existence of which **degrades measures** to protect against loss of Confidentiality, Availability, or Integrity.

1.4 STIG Distribution

Parties within the DoD and Federal Government's computing environments can obtain the applicable STIG from the Cyber Exchange website at https://cyber.mil/. This site contains the latest copies of STIGs, SRGs, and other related security information. Those without a Common Access Card (CAC) that has DoD Certificates can obtain the STIG from https://public.cyber.mil/.

1.5 Document Revisions

Comments or proposed revisions to this document should be sent via email to the following address: disa.stig_spt@mail.mil. DISA will coordinate all change requests with the relevant DoD organizations before inclusion in this document. Approved changes will be made in accordance with the DISA maintenance release schedule.

1.6 Other Considerations

DISA accepts no liability for the consequences of applying specific configuration settings made on the basis of the SRGs/STIGs. It must be noted that the configuration settings specified should be evaluated in a local, representative test environment before implementation in a production environment, especially within large user populations. The extensive variety of environments makes it impossible to test these configuration settings for all potential software configurations.

For some production environments, failure to test before implementation may lead to a loss of required functionality. Evaluating the risks and benefits to a system's particular circumstances and requirements is the system owner's responsibility. The evaluated risks resulting from not applying specified configuration settings must be approved by the responsible Authorizing Official. Furthermore, DISA implies no warranty that the application of all specified configurations will make a system 100% secure.

Security guidance is provided for the Department of Defense. While other agencies and organizations are free to use it, care must be given to ensure that all applicable security guidance is applied both at the device hardening level as well as the architectural level due to the fact that some of the settings may not be able to be configured in environments outside the DoD architecture.

1.7 Product Approval Disclaimer

The existence of a STIG does not equate to DoD approval for the procurement or use of a product.

STIGs provide configurable operational security guidance for products being used by the DoD. STIGs, along with vendor confidential documentation, also provide a basis for assessing compliance with Cybersecurity controls/control enhancements, which supports system Assessment and Authorization (A&A) under the DoD Risk Management Framework (RMF). DoD Authorizing Officials (AOs) may request available vendor confidential documentation for a product that has a STIG for product evaluation and RMF purposes from disa.stig_spt@mail.mil. This documentation is not published for general access to protect the vendor's proprietary information.

AOs have the purview to determine product use/approval IAW DoD policy and through RMF risk acceptance. Inputs into acquisition or pre-acquisition product selection include such processes as:

- National Information Assurance Partnership (NIAP) evaluation for National Security Systems (NSS) (http://www.niap-ccevs.org/) IAW CNSSP #11
- National Institute of Standards and Technology (NIST) Cryptographic Module Validation Program (CMVP) (http://csrc.nist.gov/groups/STM/cmvp/) IAW Federal/DoD mandated standards
- DoD Unified Capabilities (UC) Approved Products List (APL) (http://www.disa.mil/network-services/ucco) IAW DoDI 8100.04

2. TECHNOLOGY OVERVIEW

This section provides background information on the Microsoft .NET 4.0 Framework and discusses general security considerations involved with using this technology. This overview document is not intended as a comprehensive source of information on .NET. Microsoft and other authors have produced documentation available for reference. Additionally, this STIG is not intended as a tutorial or training tool for inexperienced SAs. Since .NET is part of an application development and runtime architecture, knowledge of how .NET applications function, the Windows OS and application development techniques and programming issues is a prerequisite to understanding how to use the .NET STIG requirements.

2.1 Introduction

The .NET Framework is an application development platform that provides services for building, deploying, and running desktop and web applications, as well as web services. It consists of two major components: the CLR, which provides memory management and other system services, and an extensive class library, which includes tested, reusable code for all major areas of application development.

The .NET Framework is divided into client and development versions. The development version is used during the application development process and the client version is utilized as a run time engine for the finished application. For the most part, the .NET Framework is completely transparent to end-users if they do not perform application development tasks within the Framework. Unless the user is developing .NET applications, the lighter client version is the desired version to install on user and host systems. .NET is designed to meet the following objectives.

- To provide a consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
- To provide a code-execution environment that minimizes software deployment and versioning conflicts.
- To provide a code-execution environment that promotes safe execution of code, including code created by an unknown or semi-trusted third party.
- To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

.NET applications are targeted to a specific .NET Framework version and require the appropriate Framework version to be installed in order to function.

2.2 .NET Framework Topology

The .NET Framework has two main components: the CLR and the .NET Framework Class Library. The CLR is the foundation of the .NET Framework and acts as an agent responsible for managing application code at execution time. The CLR provides memory management, thread management, and remoting services.

The CLR also enforces programmatic type safety, which helps to prevent erroneous or undesirable program behavior caused by discrepancies between differing data types.

Application code that targets the runtime is known as managed code, while application code that does not target the runtime is known as unmanaged code. The class library, the other main component of the .NET Framework, is a comprehensive, object-oriented collection of reusable types that can be used to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET, such as Web Forms and XML Web services.

The .NET version 4.0 guidance focuses on securing the CLR and how applications utilize the CLR rather than addressing the use of the .NET class library.

2.3 Product Dependencies

Each version of the .NET Framework including version 4.0 contains the CLR as its core component, and includes additional components, such as the base class libraries and other managed libraries. Every new version of the .NET Framework retains features from the previous versions and adds new features.

Although the CLR is the core component of the .NET Framework, the CLR is identified by its own version number apart from the .NET Framework version number. Some versions of the .NET Framework include a new version of the CLR, but others use an earlier version. For example, the .NET Framework version 4 includes CLR version 4, but the .NET Framework 3.5 includes CLR 2.0. (There is no version 3 of the CLR.)

Previous versions of the .NET Framework or the CLR do not have to be installed before installing the latest version; each version provides the necessary components.

Some versions of the .NET Framework are installed automatically with the Windows operating system, but other versions must be installed separately. Table 1-2 identifies the .NET Framework versions and whether they are integrated into the installation of Windows or if they must be installed separately.

Table 2-1: Windows Installations of the .NET Framework

.NET Framework versions	Windows versions
1.0, 1.1, and 2.0	Not installed as part of the Windows operating system, but can be installed separately on Windows XP and earlier versions of Windows.
3.0 (and 2.0 SP2, which provides support for versions 3.0 and 3.5)	Installed by Windows Vista and Windows Server 2008.
3.5 SP1	Installed by Windows 7.
4	Not installed as part of the Windows operating system, but can be installed separately on Windows XP, Windows Server 2003, and later versions of Windows.

The .NET Framework 4 is backward-compatible with applications built with previous versions of the Framework. Most applications and components built with previous versions of the .NET Framework will work on the .NET Framework 4. However, in practice, this compatibility can be broken by changes in the .NET Framework that conflict with methods employed in application programming. For example, using a hard-coded path to .NET Framework assemblies (programs), performing an equality comparison with a particular version of the .NET Framework, and getting the value of a private field by using reflection are not backward-compatible practices.

In addition, each version of the .NET Framework includes bug fixes and security-related changes that can affect the compatibility of some applications and components.

.NET Framework applications and components must be tested to ensure they are compatible with other versions of the .NET Framework before using the applications in a production environment.

2.4 Security Considerations

With the advent of .NET version 4, the security model of the Framework has changed considerably. In version 4, the .NET Framework no longer enforces security policy and the role of policy enforcement is now relegated to operating system layer components and runtime hosts.

Historically, the .NET Framework has provided code access security (CAS) policy as a mechanism to tightly control and configure the capabilities of managed code (.NET applications). Although CAS policy is powerful, it can be complicated and restrictive. Furthermore, CAS policy does not apply to native applications which are applications not controlled by the .NET CLR, so the security guarantees offered by CAS policy are limited.

In .NET Framework version 4, machine-wide security policy is turned off by default. Applications that are not hosted (that is, applications that are executed through Windows Explorer or from a command prompt) now run as full trust. This includes all applications that reside on shares on the local network. Hosted or sandboxed applications continue to run with trust policies that are decided by their hosts (for example, by Internet Explorer, ClickOnce, or ASP.NET). Applications or controls that run in sandboxes are considered partially trusted.

To control the execution of applications, dll libraries, and scripts, system administrators must now use tools that function at the operating system layer rather than utilizing .NET Framework tools, such as CASPOL.EXE or MSCORCFG.EXE. Microsoft provides two tools to control application execution at the operating system level; they are Software Security Policies and AppLocker. Both of these tools are implemented via Group Policy.

The transparency model has also been applied to the .NET Framework. Applications and controls that run in a host or sandbox with the limited permission set granted by the sandbox are considered transparent. Transparency means there does not have to be concern about checking CAS policy when running partially trusted applications. Transparent applications just run using their grant set.

By utilizing the transparency model, sandboxing methods, Software Security Policies and AppLocker, security protection capabilities are enhanced to address .NET applications, as well as native code applications that run outside the protections offered by the CLR.

3. SECURITY READINESS REVIEW (SRR)

3.1 SRR Overview

The .NET Framework Security Readiness Review (SRR) targets conditions that undermine the integrity of security, contribute to inefficient security operations and administration, or may lead to interruption of production operations. Additionally, the review ensures the site has properly installed and implemented the .NET environment and it is being managed in a way that is secure, efficient, and effective. The items reviewed are based on the NSA guide, *Guide to Microsoft .NET Framework Security* and vendor recommendations provided by Microsoft. The results of the review should be recorded in the SRR Results section with the following status designations: F- Finding, N/F- Not A Finding, N/A- Not Applicable, MR- Manual Review, or NR- Not Reviewed. The items reviewed are based on standards and practices published by Microsoft® (the vendor) and other security guidance entities, following guidance published in the Department of Defense Instruction (DoDI) 8500.2 and National Institute for Standards and Technology (NIST) Special Publication (SP) 800-53 security controls.

Defense Information Systems Agency (DISA) Field Security Operations (FSO) has assigned a level of urgency to each finding based on Chief Information Officer (CIO)-established criteria for Certification and Accreditation (C&A). All findings are based on regulations and guidelines. All findings require correction by the host organization.

3.2 SRR Review Method

To perform a successful SRR, this document and accompanying STIG provide the methods to assess vulnerabilities on systems running Microsoft® .NET 4.0. To perform a successful SRR, a manual process must be employed. There are currently no automated tools to check for compliance with this checklist.

Since each version of the .NET Framework is configured separately, an SRR must be performed against each version of the .NET Framework that is installed on the system. This guidance pertains to .NET Framework version 4.0 only. When conducting an SRR on a previous version of the .NET Framework, refer to existing guidance for instructions that address previous versions of the Framework.

3.3 SRR Additional Considerations

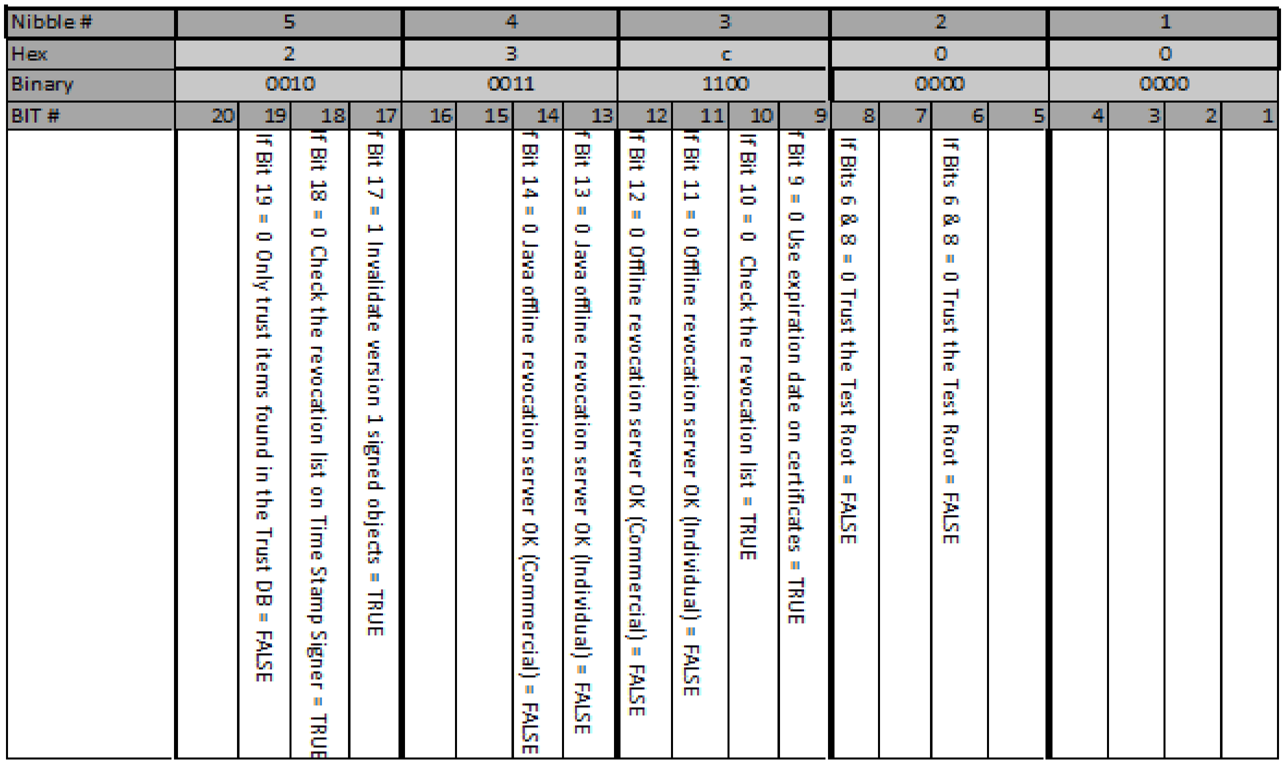
To create these STIG requirements, the principles and guidelines found in the DoDI 8500.2 IA controls were applied to version 4.0 of the .NET Framework. The .NET 4.0 configuration tested was a basic default installation without third party applications installed or other enhancement.

3.4 SRR .NET Software Publishing Table

The following table may be used during the SRR process as a guide to help in understanding the overall functionality of the Windows Software Publishing Table. The functionality displayed in the table columns is configured via a Windows registry setting. The hexadecimal value set in the Windows registry directly affects system behavior relating to Windows Authenticode.

Authenticode is a Microsoft technology provided with the Windows OS that is used to validate signed application certificates.

Table 3-1: Software Publishing State



Nibble #	5				4				3				2				1			
Hex	2				3				c				0				0			
Binary	0010				0011				1100				0000				0000			
BIT #	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
		If Bit 19 = 0 Only trust items found in the Trust DB = FALSE	If Bit 18 = 0 Check the revocation list on Time Stamp Signer = TRUE	f Bit 17 = 1 Invalidate version 1 signed objects = TRUE			f Bit 14 = 0 Java offline revocation server OK (Commercial) = FALSE	f Bit 13 = 0 Java offline revocation server OK (Individual) = FALSE	If Bit 12 = 0 Offline revocation server OK (Commercial) = FALSE	If Bit 11 = 0 Offline revocation server OK (Individual) = FALSE	If Bit 10 = 0 Check the revocation list = TRUE	f Bit 9 = 0 Use expiration date on certificates = TRUE	If Bits 6 & 8 = 0 Trust the Test Root = FALSE		If Bits 6 & 8 = 0 Trust the Test Root = FALSE					

APPENDIX A - GLOSSARY OF COMMONLY USED .NET TERMS

Application domain: The logical and physical boundary created around every .NET application by the Common Language Runtime (CLR). The CLR can allow multiple .NET applications to be run in a single process by loading them into separate application domains. The CLR isolates each application domain from all other application domains and prevents the configuration, security, or stability of a running .NET application from affecting other applications. Objects can only be moved between application domains by the use of remoting.

Assembly: All of the files that comprise a .NET application, including the resource, security management, versioning, sharing, deployment information, and the actual MSIL code executed by the CLR. An assembly may appear as a single DLL or EXE file, or as multiple files, and is roughly the equivalent of a COM module.

Code Access Security (CAS): A mechanism provided by the CLR whereby managed code is granted permissions by security policy and these permissions are enforced, helping to limit the operations that the code will be allowed to perform. This model was changed in .NET 4.0 and policy is no longer enforced in the Framework.

Common Language Runtime (CLR): The engine at the core of managed code execution. The runtime supplies managed code with services, such as cross-language integration, code access security, object lifetime management, and debugging and profiling support.

Common Language Runtime Host (Runtime Host): An unmanaged application that uses a set of APIs, called the hosting interfaces, to integrate managed code into the application. Common language runtime hosts often require a high degree of customization over the runtime that is loaded into the process. The hosting interfaces allow common language runtime hosts to specify settings that configure the garbage collector, select the appropriate build for their environment (server versus workstation), and so on. Common language runtime hosts often support an extensibility model that allows the end user to dynamically add new pieces of functionality, such as a new control or a user-written function. These extensions are typically isolated from each other in the process using application domains and custom security settings. Examples of common language runtime hosts include ASP.NET, Microsoft Internet Explorer, and a host to run executables launched from the Windows Shell. See also: application domain, common language runtime, managed code.

Configuration File: An XML file with the .config extension that contains option settings for an application or web site. Common configuration files include Machine.config, Web.config and "ApplicationName".exe.config where "ApplicationName" is a variable value representing the name of the executable file.

Evidence: Evidence is information about an assembly, such as a digital signature or the zone or site of its origin. Evidence may be contained in the assembly itself or may be presented by the host. There are currently seven types of evidence in the .NET Framework. These evidence types are:

- Application Directory – The directory where the assembly resides.
- Hash – A cryptographic hash of the assembly.
- Publisher – The publisher of the application, based upon Authenticode signing of the assembly.
- Site – The site where the assembly originated. This is only valid when the assembly is executed directly from the site.
- StrongName – A cryptographic signing of the assembly.
- URL – The URL where the assembly originated. This is only valid when the assembly is executed directly from the URL.
- Zone – The Internet Explorer Security Zone associated with the site of origin for the assembly.

Managed Code: Code executed by the common language runtime environment rather than directly by the operating system. Managed code applications gain common language runtime services, such as automatic garbage collection, runtime type checking and security support, and so on. These services help provide uniform platform- and language-independent behavior of managed-code applications. See also: unmanaged code.

Native code: Code that has been compiled to processor-specific machine code.

Remoting: The process of communication between different operating system processes, regardless of whether they are on the same computer. The .NET Framework remoting system is an architecture designed to simplify communication between objects living in different application domains, whether on the same computer or not, and between different contexts, whether in the same application domain or not. See also: application domain.

Security Policy: The active policy established by the administrator that programmatically generates granted permissions for all managed code based on the code's requested permissions. Code that requires more permissions than policy will grant is not allowed to run.

Unmanaged Code: Code that is executed directly by the operating system, outside the common language runtime environment. Unmanaged code must provide its own garbage collection, type checking, security support, and so on, unlike managed code, which receives these services from the common language runtime. See also: Managed Code.