Security in the Microsoft .Net Framework

Contents

1. **Security in the Microsoft® .NET Framework** **2**
2. **Understanding Dot NET** 2

2.1 Basic Security Points 2

2.1.1 Encryption 2

2.1.2 Hashing 3

2.1.3 Digital Signatures 3

2.1.4 Digital Certificates 4

2.1.5 Secure Communication 4

3. **.NET Framework Security in Detail** 5

3.1 Evidence-based Security 5

3.1.1 Policy 5

3.1.2 Permissions 5

3.1.3 Isolated Storage 5

3.1.4 Evidence 6

3.2 CAS – Code Access Security 6

3.2.1 Security Check Stack Walk 6

3.3 The Verification Process 6

3.4 Role-Based Security 7

3.4.1 Authentication 7

3.4.2 Authorization 8

3.4.3 Principal & Identity 8

3.4.4 Cryptography 09

3.4.5 Application Domains 09

White Paper | Security in the Microsoft .Net Framework

1. Security in the Microsoft® .NET Framework

Creating quality, secure software products is not easy. As major software developers are facing major challenges on ensuring secured code, this document presents the security measures developed by Compvue to protect all of our applications. When building applications, Compvue engineers adopt every possible security measure to ensure maximum security. The Microsoft® Dot NET Framework provides developers with a powerful set of tools to ensure application security. This document aims to provide an implementation overview as well as information on our security practices.

2. Understanding .NET Security

In this latest .NET framework, two major changes have been implemented to enhance security and simplify design. Though the permissions system remains in place, the Dot NET 4.0 framework has eliminated its machine-wide security policy and has made security transparency its default mechanism. Figure 2.1.1 offers a brief overview of this framework.

2.1 Basic Security Points

Before we expand on deeper topics, it is important to understand more basic security points like Encryption, Hashing, Digital signatures, Digital certificates, Secure Communication, Authentication, Authorization, Firewalls, Auditing, Service Packs, and updates.

2.1.1. Encryption

Encryption protects sensitive data from being viewed or modified and ensures secure channels of communication. We can encrypt data using a cryptographic algorithm, which is then transmitted in an encrypted state.

Algorithm	Description
Symmetric	**Uses one key to:** - Encrypt Data - Decrypt Data **This method is fast and efficient.**
Asymmetric	**Uses two mathematically related keys:** - Public key to encrypt the data. - Private key to decrypt the data. **This method is more secure than symmetric encryption, but is also slower.**

This can later be decrypted by the intended party and ensures protection against theft from third parties. We can further use this methodology to ensure:

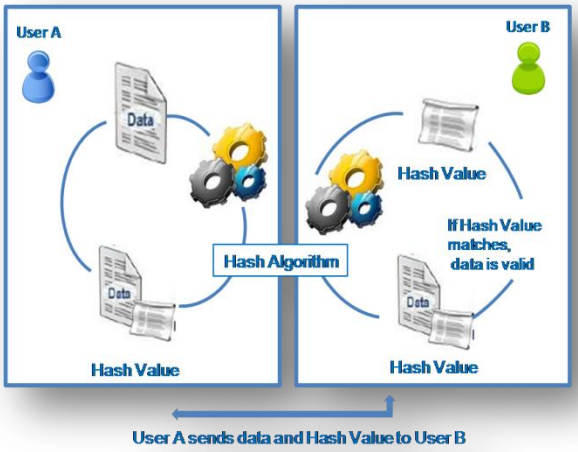
- **Confidentiality** – Protects PII data of a user.
- **Data Integrity** – Protects data from being altered from unauthorized members.
- **Authentication** – Verifies that data originates from a particular party.

Symmetric: A symmetric encryption transforms the data to prevent unauthorized third parties from accessing it. It uses a single shared, secret key to encrypt and decrypt data. This type of data transfer is fast and consumes a fewer amount of resources. Common algorithms for this type include: Data Encryption Standard (DES), Triple DES, Advanced Encryption Standard (AES), international Data Encryption, Algorithm (IDEA) & RC2.

Asymmetric: In an asymmetric encryption method, a pair of keys is used to encrypt and decrypt the data. This process consists of two parts – a public and private key that are mathematically related. This method is far more secured than a symmetric encryption, but consumes more

resources as well. Common algorithms for this type include: Rivest, Shamir and Adleman (RSA), Diffie-Helman, and Digital Signature Algorithm (DSA).

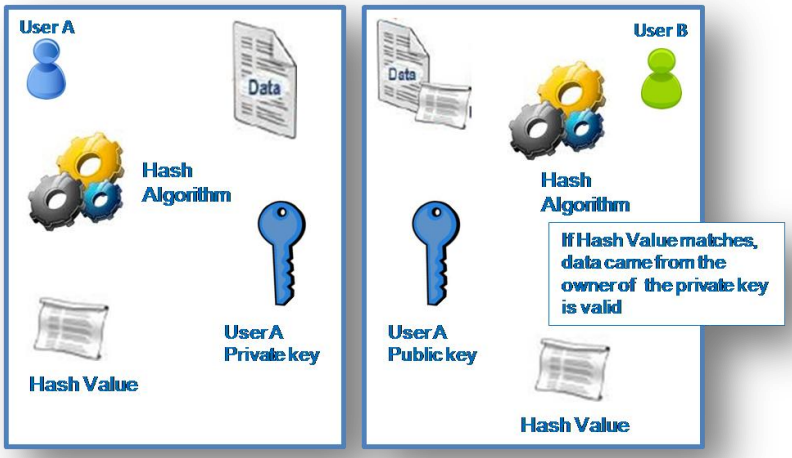
2.1.2 Hashing



Hashing differs from encryption in that encryption is the process of encoding data for later decryption, while hashed values are validated and verified for data integrity. A hashed code is a unique, fixed-length string of bits. The process of hashing involves matching an arbitrarily long string of bits with its fixed-length counterpart. Has processes should be both collision resistant and one-way operations. Therefore, it must be computationally unfeasible to determine the input data from just the hash value. Please refer to the below diagram where "User A" sends the hash value with original data.

"User B" verifies the validity of this data by applying the same hash algorithm and then comparing the resulting value to the original hash value sent by User A. If the values match, User B can be assured that nobody has tampered with the data since it was first sent. Common hash functions include: Message Digest 4 (MD4), Message Digest 5 (MD5) & Secure Hash Standard (SHA-1).

2.1.3 Digital Signatures

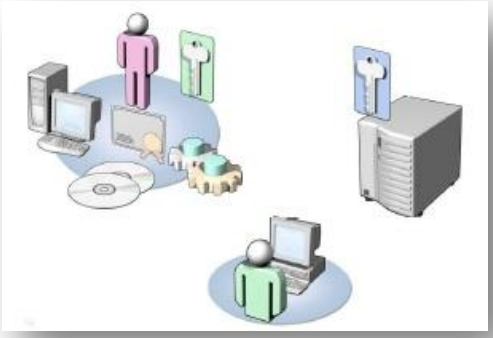


Digital signatures ensure that data originates from a specific user by creating a signature unique to that user. Digital signatures combine hashing and asymmetric encoding for the respective validation and encoding of signature data. The below diagram explains how and when data is signed with a digital signature.

The digital signature method applies a hash algorithm to the data to create a hash value. The hash value is then encrypted with User A's private key, which creates the digital signature. The signatures and data are then sent to User B. User B can decrypt the signature and then recover the hash value by using User A's public key. If the signature can be decrypted, User B knows that the data came from User A (the owner of the private key). User B then applies the hash algorithm to create a second hash value that should match with User A's (if the data has not been tampered with).

White Paper | Security in the Microsoft .Net Framework

2.1.4 Digital Certificates



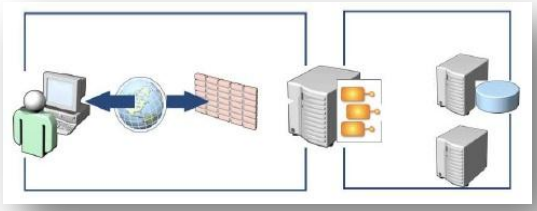
A digital certificate improves on digital signatures by allowing the identification of a party. For example, if an attacker obtained a private key belonging to Microsoft, they could easily send corrupted data using that key and a standard hashing algorithm since the source of the data belongs to Microsoft. Digital certificates solve this problem by verifying that a signature does indeed belong to its publisher. Both data and signature can be verified as belonging to the publisher.

The user, computer, and/or service/application creates a public/private key pair. The public key is transmitted to the certification authority (CA) through a secure network connection. The certified administrator then reviews the certificate request to verify the information. Based on approval, the CA signs the public key with the private key of the CA. Commonly used methods include 802.1x wireless communication, digital certificates, encryption files system (EFS), internet authentication, IP security (IPSec), Smart card logon, Software code signing, & software restriction policy.

2.1.5 Secure Communication

Secure communication ensures the privacy and integrity of data across a network by using the below technologies.

IPSec – Internet Protocol Security: IPSec is a framework of open standards that ensure secure, private communication over IP networks. This system uses a combination of cryptographic security devices that are negotiated between the client and server as well as encryption standards which include symmetric algorithms like DES, Triple DES, and RC5. IPSec also uses hashing processes such as MD5 and SHA-1 and provides a transport-level secure communication that can be used to secure data that is sent between two entities (i.e. an application server and database server).



SSL – Secure Sockets Layer: Every request or response is validated by the authority of the certificate. The certificate is passed to the application with a request.

TLS – Transport Layer Security: This method is used whenever a server needs to verify its identity for anonymous clients. For example, an e-commerce application should ensure the secure transmission of sensitive information like credit card details. This mandates that all servers prove their identity to clients. TLS also provides options for clients to prove their identities to servers. The TLS method is the ideal option for environments where we require the highest level of security for the data in transit.

RPC – Remote Procedure Call: Using the Distributed Component Object Model (DCOM), RPC protocol provides packet-level privacy authentication. Every packet of data sent between client and server is then encrypted. The RPC runtime stubs and libraries take care of details relating to network protocols and communication. This helps us focus on details related to the application rather than the network.

3. .NET Framework Security in Detail

The .NET framework security system is a part of the Common Language Runtime, or CLR. This system includes many features such as checking for safe type-conversions, secure exception management, and code access security control. This .NET framework security is designed in such a way that it complements security features available on Microsoft Windows though it doesn't override any of the Windows-based security mechanisms [ex: if a Windows access control list (ACL) restricts access to a file, it does not override its security].

The .NET framework security architecture includes:

- Evidence-based security
- The verification process
- Cryptography
- Code access security
- Role-based security
- Application domains

We will now explore each aspect in more depth.

3.1 Evidence-Based Security

The key elements of the .NET Framework evidence-based security subsystem include policy, permissions, and evidence.

3.1.1. Policy

The .NET framework security rests on carefully defined, XML-inscribed policy. It ensures data integrity by preventing software from harm from non-administrative user access. Policy can be set on every machine and for each user account with the help of the Windows domains Group Policy. This ensures a safe execution environment for every user.

3.1.2. Permissions

Permissions refer to one or more resources and related rights. These can be related to DNS, Data Access, Directory Services, Folio, Event Log, Environment, File Dialog, Registry, Reflection, Socket, Web, Isolated Storage, UI, Printing, Message Queue, and Security.

3.1.3. Isolated Storage

Isolated Storage helps in implementing special file storage mechanisms, which makes secured operations related to files. The data storage mechanism provides isolation and safety by defining standardized ways of associating code with saved data. It also isolates different applications repositories and any specific file system characteristics like path names. This helps to store

application state and user preference information and helps developers from not creating unique paths to specify safe locations in the file system. Developers can access safe locations by using the application's identity or the user's identity.

3.1.4. Evidence

CLR evaluates an assembly's evidence during runtime and determines which permissions can be assigned to a particular assembly. This evidence can be obtained from sources resident within an assembly or can be identified from the local execution environment. The evidence sources are cryptographically sealed namespaces (strong names), software publisher identity (Authenticode®), and code origin (URL, site, Internet Explorer Zone).

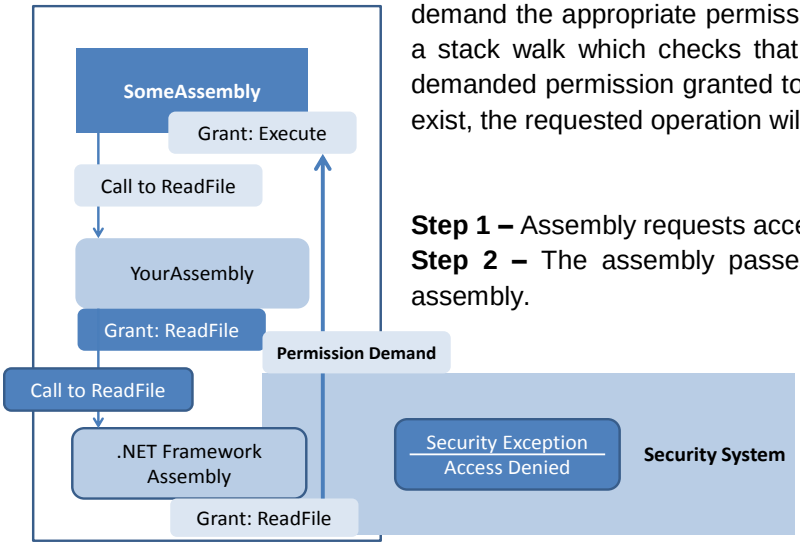
3.2 CAS – Code Access Security

Code access security (CAS) is the enforcement engine that makes sure that the assembly code has restricted its granted permissions while executing the source code within the computer system.

The Dot NET Framework security policy is built using an assembly identity, rather than a user identity. Dot Net security works on top of Win32 security (and is thus not a replacement to Win32). The framework also collects evidence about an assembly and presents it to the security system after which the runtime decides whether or not to allow the code to complete all of the tasks that it requests. It also requires developers to create their own unique evidence.

3.2.1. Security Check Stack Walk

Whenever an assembly requires permission to access a resource, the code providing access will demand the appropriate permission object. The demand works by initiating a stack walk which checks that each assembly in the call-chain has the demanded permission granted to the object. In case a permission does not exist, the requested operation will not be performed.



Step 1 – Assembly requests access to a method in an assembly.
Step 2 – The assembly passes the request to a Dot NET Framework assembly.

Step 3 – A Security system validates whether all callers in the stack have the required permissions.
Step 4 – The security system either grants access or throws an exception.

3.3 The Verification Process

At the time of a JIT compilation, the CLR verifies all managed code for memory type safety. This procedure eliminates the risk of the code executing "unexpected" actions. Otherwise, it would result in bypassing the common application flow.

The verification process prevents common errors such like accessing arbitrary memory locations, type safety, object boundary validation, member visibility restrictions, buffer overflows, entry points, and stack frames. This process helps avoid security vulnerabilities.

3.4 Role-Based Security

Whenever anyone tries to access a resource, the Dot NET framework creates a user identity and either permits or denies that identity resource access. These processes are executed using authentication and authorization type methods. The identity has information about the user's identity and uses a log-on name and user authentication status. A principal contains the role membership information of a computer user. The Dot NET Framework implements two types of identities and principals, which are **WindowsIdentity** and **WindowsPrincipal** objects respectively. These objects provide information about the Windows credentials for a user whereas **GenericIdentity** and **GenericPrincipal** objects enable the developer to implement their own authentication techniques.

3.4.1. Authentication

Compvue employs a customized authentication method using Role-Based security. The authentication uses request contexts by verifying credentials and creates proper Identity and Principal Objects. An authorization decision is made after determining the user identity for accessing the resources. This includes:

- **Forms-based (Cookie) Authentication:** Using this provider causes unauthenticated requests to be redirected to a specified HTML form using client-side redirection. The user can then supply log-on credentials and post the form back to the server. If the application authenticates the request (using application-specific logic), ASP Dot NET issues a cookies that contains the credentials, or a key, for reacquiring the client identity. Subsequent requests are issued with the cookie in the request headers, which means that subsequent manual authentications are unnecessary. The credentials can be custom checked against different sources, such as an SQL database or a Microsoft Exchange directory. This authentication module is often used when the developer wants to present the user with a log-on page.
- **Passport Authentication:** This is a centralized authentication service provided by Microsoft that offers a single logon facility and membership services for participating sites. ASP Dot NET, in conjunction with the Microsoft Passport Software Development Kit (SDK), provides functionality similar to Forms Authentication for Passport users.
- **IIS:** Microsoft's IIS server provides several built-in authentication mechanisms which can be used to provide authenticated identities to IIS-hosted applications. If there are corresponding Windows accounts, IIS can also provide automatic account mapping based on the authenticated identity. Supported authentication mechanisms include Basic Authentication, NTLM, Kerberos, Digest Authentication, and X.509 Certificates (with SSL).
- **Windows Authentication:** Windows supports a number of authentication mechanisms that can be used by an application via the SSPI subsystems. These include Kerberos, NTLM, and X509 certificates. Developers can also write custom authentication and authorization code (for example, by combining IIS Anonymous authentication with ASP Dot NET's form authentication provider), or use one of the standard authentication modules already available in the ASP Dot NET framework (by combining IIS NTLM or Kerberos authentication with ASP Dot NET's Windows authentication provider). Authentication providers can be also configured for each application and virtual directory.

3.4.2. Authorization

Once an identity is established reliably using one of the above methods, access to resources can be authorized through a similarly extensible and flexible architecture. ASP Dot NET provides two different methods of authorization to application code:

- File Authorization, where the request location is mapped to a physical file, denying or granting access by matching the file's ACLs with the request-making identity.
- URL Authorization, where access can be granted or revoked by specifically mapping users and roles to pieces of the URI namespace, including the request method (GET, HEAD, POST, etc).

For example, to restrict access to the URL "http://servername.com/adminpage.aspx" to users in the role "Admin," one could perform the above runtime role checks in code.

3.4.3. Principal and Identity

The Dot NET Framework provides a rich and robust object model for identities using its Principal and Identity concepts. A Principal represents the security context under which the code is running, while an identity represents the identity of the user associated with that security context. Normally, an Identity will be created after a user's successful authentication, and will be attached to a Principal that will in turn be associated with an execution context. Code running in a specific context can then query the Principal about the Identity role(s), allowing or denying permissions according to the role membership.

This architecture is flexible enough to permit custom definitions of roles, identities, and principals. For example, it is possible to map identities to username/password pairs stored in a database or text file. Implementing the GenericPrincipal objects allows for these highly customized, platform-independent authorization scenarios.

Alternatively, the Dot NET framework can leverage traditional Windows security subsystems via the Windows Principal objects, allowing the easy mapping of roles to existing Windows user accounts and groups.

Of course, the Dot NET framework is capable of performing impersonations of client requests to access resources. Impersonation remains one of the key differentiators between Windows-based authorization architectures and competitive solutions like UNIX and Linux. This allows solutions architects to keep an identity tied to one user account throughout the flow of an application, rather than periodically handing off control to the process under which the application runs.

Impersonation in the ASP Dot NET framework can be implemented in two different ways:

- Per-request impersonation, which means that an application can run with the privileges of the identity making the request. This helps in reducing the impact of possible security breaches while improving auditing capabilities.
- Application-level impersonation, where the worker process running the application does so using the identity of a user specified in the configuration, diminishing the impact of application compromise by isolating and protecting other applications sharing the same server and system (i.e. application compromise doesn't necessarily lead to system compromise).

Impersonation gives ASP Dot NET applications granularity and flexibility when accesing resources, homogeneously across the Dot NET framework.

3.4.4. Cryptography

Similar to the ready availability of simple authentication and authorization features within the Dot NET framework, cryptographic primitives are also easily accessible to developers via stream-based managed code libraries for encryption, digital signatures, hashing, and random number generation. Wrappers for most CryptoAPI functionalities are also available. Algorithm support includes:

- RSA and DSA public key encryption (asymmetric)
- DES, TripleDES, and RC2 private key encryption (symmetric)
- MD5 and SHA1 hashing

Besides the supported primitives, the Dot NET framework supports encryption by means of cryptographic streaming objects based on the implemented primitives and various feedback modes. It also supports digital signatures; message authentication codes (MAGs); keyed hash; pseudo-random number generators (PRNGs); and authentication mechanisms. New or pre-standard primitives such as SHA-256 or XMLDSIG are already supported. ASP Dot NET includes well-integrated support for signing and encrypting cookie content addressing long-standing sensitive issues of Web application security.

The ready availability and more than complete breadth of such libraries will hopefully drive more widespread reliance on the cryptography to fortify the security of everyday applications. Based on our own experiences, we can confidently state that well-implemented cryptography dramatically increases the security of many aspects of a given application.

3.4.5. Application Domains

The Dot NET framework offers a compelling new way to segregate portions of applications through what is known as application domains. Usually, operating systems provide this isolation by running each application in a separate process, each one having a different address space. This prevents these applications from directly interfering with each other. Unfortunately, for highly loaded servers, processes are expensive in terms of system performance and it may be prohibitive to run an individual process for each user that is accessing the server.

Thanks to the type-safety of verified managed code (which ensures, among other things, that the code cannot access or jump to arbitrary addresses in memory), the CLR is able to provide a great level of isolation within the process boundary. A single process can contain several application domains, with different evidence-based trust levels and associated principals, without danger to any kind of malicious interference between them. Code running in one domain cannot directly affect other applications in the same process, or access other application resources. All managed code is loaded into a single application domain and run according to that domain's security policy.

All in all, application domains are a tremendous boon for Application Service Providers and IT departments hosting networked applications. They offer powerful security controls at a fraction of the resource costs of existing solutions.