Global Information Assurance Certification Paper

MICROSOFT .NET FRAMEWORK SECURITY

September 2002
Raymond Ng

Abstract: *This article takes a look at Microsoft's .NET Framework and how it attempts to improve on security for the end user, administrator and developer.*

Table of Contents

Scope of this document

This document will cover the security aspects of the .NET Framework . This document will avoid comparisons to other competing products , except where necessary to aid clarification, and will concentrat e on informing the reader on the security benefits and issues of the .NET Framework.

This document is targeted for the technical audience – that wish to understand what .NET security is about and how it achieves it on a technical level.

What is the Microsoft .NET Framework?

According to Microsoft '.NET is a set of software techn ologies designed to connect the world of information, people, systems, and devices '[i] and that the .NET Framework is the 'programming model that enables developers to build Web-based applications, smart client applications, and XML Web services applications which expose their functionality programmatically over a network using standard protocols such as SOAP and HTTP. '[ii]

Before we can delve into the technical a spects of the .NET Fram ework it's helpful to understand the three basic components of the Framework. These are: The Common Language Runt ime, Class Libraries and Assemblies.

Common Language Runtime (CLR)

The CLR is the Virtual Engine that is responsible for executing code. The C LR must police everything that is run and must also enforce restrictions on any code that behaves unexpectedly or outside its security boundaries. All code run within the CLR is known as *Managed Code*.

The CLR is 'big brother' when it comes to security sin ce it implements its own secure execution model that is independent of the host platform. This means that it introduces a secure runtime environment to platforms that have never had it, such as Windows 98.

Class Libraries

The Class Libraries are a collect ion of reusable classes which can be used and extended by developers when creating .NET applications. These classes are native to the Framework and implement many important security features, such as:

- Permissions (FileIO, Environment, etc)
- Authentication
- Cryptography

Assemblies

An *assembly* is a program (executable or DLL) that is complied into code that the CLR understands. The .NET Framework has compilers for most of today's major programming languages such as, C++, C#, Java, COBOL, Visual Basic, etc.

The Framework's compilers translate the programming language into Microsoft Intermediate Language (MSIL) , which are the actual *assemblies*. This gives developers the freedom to program in their preferred programming language and still have their code inter-operate with all other *assemblies*.

What does .NET offer in security for end users?

The Situation

Up to now Microsoft's approach to security for the home user was simply 'put your faith in the company who wrote the software. ' All too often are users prompted to install, execute and trust an ActiveX signed control and many users do so without even reading the warnings in the prompts.

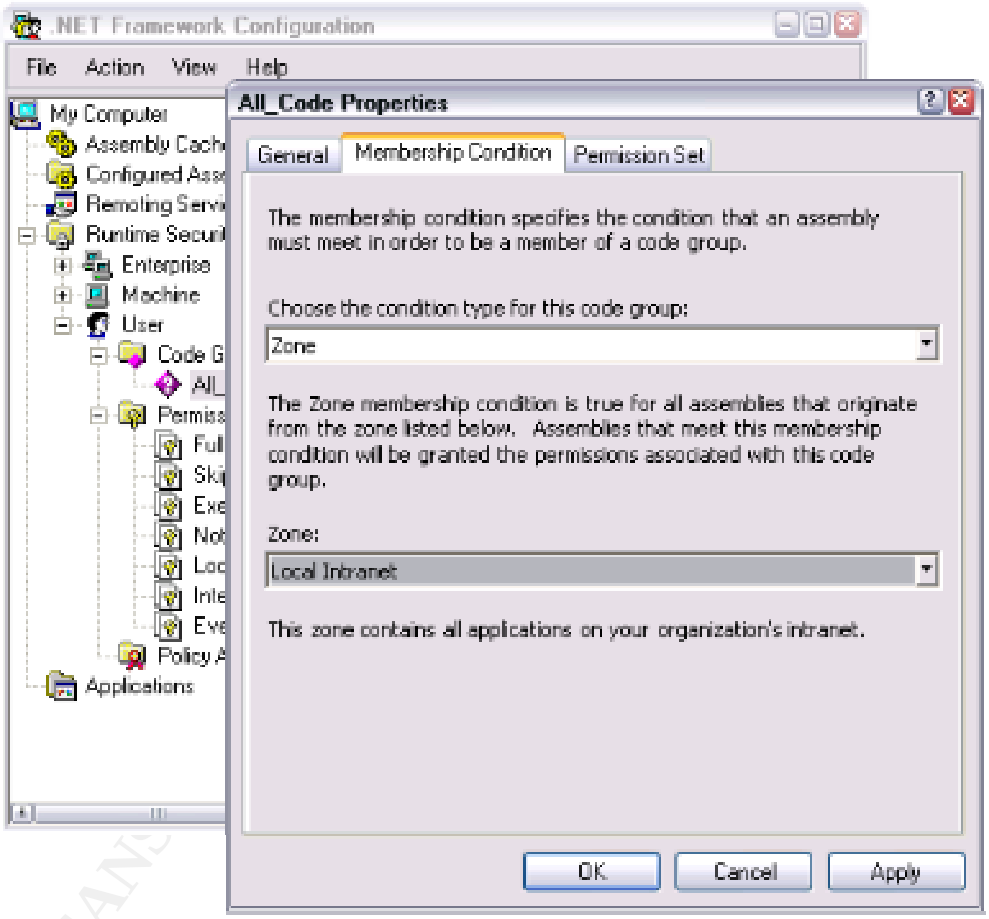
The alternative was to configure the browser to disable the use of ActiveX controls altogether. This, however, could limit the experience and functionality that the user could have.

Evidence-based security

The .NET Framework determines what permissions should be granted before any code is run. This is known as the code's *evidence*. Evidence can be anything and everything that the system knows about the code. Source of evidence include[iii]:

- Cryptography sealed namespaces (strong-named assemblies)
- Software publisher identity (Authenticode)
- Code origin (URL, site, Internet Explorer Zone)

The permissions that any code will be grante d is based on the Security policy.



These permissions can be set by the user or by the administrator. The default security policy installed with the .NET Framework was designed by Microsoft intended for the average user.

There are a number of configura ble policy levels for the Security policy [iv]:

- Enterprise Policy
- Machine Policy
- User Policy
- Application Domain Policy

The enterprise, machine and user policies are configurable by security policy administrators. The application domain policy is configurable, programmatically, by the host. The resulting permissions are an intersection of all evidence that is available and all policies which apply to it.

After ensuring that the code is granted the minimum permissions specified by the security policy, the Frame work determines which permissions the assembly requires in order to execute and excludes all permissions that are not required. For example, if a program does not need any *FileIO* permissions, these permissions will not be granted when the assembly is execu ted.

If the assembly requires more permissions than has been granted then it will fail to run.

Because the code is examined to determine which permissions are required, any all other permissions are no granted. The .NET Framework has the ability to prevent code from executing in any unexpected way. An example of this is if the developer of the code does not specify that the assembly needs the permission to list the local system's directory and files, then the .NET Framework will prevent this action from t aking place.

It is worth noting that the any code executed from the local hard disk which, by default, is much more trusted than code that is executed from any other remote locations (Internet/Intranet). So in this scenario the burden is left on the user of the system to know the difference between code run locally and code run remotely.

Code Access Security (CAS)

The above section talks about how permissions are granted to any code that is executed by the CLR. However there may be situations where one as sembly uses one or more other assemblies to perform tasks. Assemblies which might be granted more permissions. A hacker may try to write some code that tries to 'trick' another assembly in the attempted to bypass security.

Fortunately when such an event o ccurs a *stack walk* takes place. This process checks that each assembly in the call -chain has to the appropriate permissions to perform the function. Note that this does not limit developers since they are able to override portions of the stack walk operation. This is investigated further in the section titled 'declarative and imperative security .'

Verification

Finally, most managed code is verified to ensure type safety as well as the well - defined behaviour of other properties. This prevents many errors th at occur by either a developer who has missed a bug or a hacker wishing to exploi t a weakness in the application. To illustrate how the CLR operates, take a variable that is allocated a 4 byte of memory space, the CLR will prevent any attempts to write any value of more than 4 bytes to this space , thus protecting the

possibility of overwriting the allocated memory space and into another address outside the bounds of the current program. This is more commonly known as a causing a buffer overflow error. In addition the CLS used *verification* to ensure that a pointer does not access memory that has not been allocated to the program, therefore ensuring that execution will flow onto to well known locations.

This is a very powerful feature of the .NET Framework and means that many of the common programming errors do not lead to comprised of the system. Common vulnerabilities such as buffer overruns and the reading of memory of or bounds are no longer a threat in the safe verified environment of the framework.

Amazingly, one program can consist of many assemblies, each of which the CLR will test separately for its evidence, code-access behaviour and will also be put through the verification process. This locks each individual assembly in its own environment. Even though it could actually be residing in the same CLR as other assemblies it cannot access other assemblies or behave in any way not specified by its metadata. (Refer to 'security for free' below).

What does .NET offer in security for developers?

The previous section shows many security advantages of the .NET Framework and its CLR engine. However it is just as important for developers to have access to unified, simple, supported and proven security mechanisms, modules and objects.

Luckily the .NET Framework comes with all of the above and still offers developers the freedom to use security standards that are not from Microsoft.

This section will go though some of the features given to developers to aim and encourage them write more secure programs.

Security for free

Just by using the .NET Framework class libraries the developer automatically inherits some of the security features available. This sounds almost to amazing to be true, but if the out-of-the-box classes are used for such events like reading and writing files and accessing environmental variables the code generate by these classes are type aware and tell the CLR what permissions are required for the code to run.

In a section above this article talks about Code Access Security and Evidence-based security. These both rely on the assembly itself informing the CLR on what permissions are needed in order for the code to work. When compiling .NET applications this data is stored with the resulting assembly as metadata. The metadata contains, amongst other things, type data for type-safe verification and what methods it implements from other classes or interfaces.

The metadata is known as a 'contract' since it is an agreement of how the assembly will access types, classes, interfaces and memory.

The information in the metadata is used to determine if the assembly is type - safe and how the CLR can expect it to behave. Anyth ing outside the behaviour of its 'contract' causes the CLR to abort the program and alert the user of a Code Access violation.

Declarative and Imperative Security

Developers are given two methods to modify the run -time security requirements of their assemblies. These are *declarative* and *imperative*.

As the name suggests declarative security enables programmers to specifically make request for the permissions required. These permission requests become part of the assembly's metadata which the CLR uses for *evidence* and *verification*.

Declarative security is useful for static like security requests, when an assembly requires specific permissio ns. A good example of this is if we needed to write to a temporary folder, eg. C: \Temp. If this permission is denied then the assembly can gracefully exit.

Declarative security can also be used to check if other assemblies have strong names or if they are signed by a specific publisher.

Imperative security is much more polymorphic and should be used where the security requirements of an assembly change during run -time or are unknown. This type of security must be implemented directly in code.

With imperative security when a assembly needs to be granted one or more permissions it uses a method called demand(). This method is used to ask other modules in the security stack the use permissions that the calling assembly does not have. For example if a program mer had to interface with a assembly that was responsible for reading data from a database. The CLR would not normally allow for this permission. The interface assembly would receive the request and can either grant or deny based on its own programmatic ch ecks.

Imperative security is implemented in code and allows for dynamic security requirements. Unlike declarative security where specific security requirements are needed, imperative security can catch failed demand requests and handle them dynamically an d programmatically.

The section title 'code access security' talks about a function of the Framework called the *stack walk*; where an assembly could not trick the CLR for more permissions by interfacing with another assembly with greater permissions. Imperative security can modify the stack trace, allowing additional permissions to be granted. Although this is a necessary feature of the Framework it allows a

easy for a hacker in if this assembly, which demand() is called upon, isn't developed with security in mind. Such assemblies must be careful who it grants permissions to and in .NET this is responsibility is wholly on the developer.

Strong-Named Assemblies

A *strong-named assembly* is when an assembly is glued to a digital signature, version number, all d igitally signed with an encrypted checksum (Private Key). This ensures that if even if one byte of the strongly named assembly (include all those things that are *glued* to it) the CLR will detect this instantly and will not allow any code to be run .

It is not advertised by Microsoft as a anti -tampering security feature but strong -named assemblies would give traditional viruses a very hard time since it is no longer as simple as ensuring that the file size or checksum match the original.

Although most prog rams don't even bother checking the checksum of it self or it components, since this normally requires additional work from the developer. The Framework makes it really simple to apply strong names to any assembly (at least within Visual Studio anyway). Th e .NET Framework SDK comes with a program to generate private keys for signing assemblies: sn.exe. Once you have generated the private key there are two attributes that need to be set in code, these are AssemblyVersion and AssemblyKeyFile. The following illustrates an example:

```
<Assembly:System.Reflection.AssemblyVersionAttribute ("1.0.0.1")>
<Assembly:System.Reflection.AssemblyKeyFileAttribute ("MyKey.snk")>
```

Strong Names in the .NET Framework have many uses. From the example above you will notice that str ong-named assemblies include a version number. This is useful for applications that use common external DLL's. When an assembly references a strong -named assembly the developer can specify which version of the assembly is expects. If this version is not av ailable when the program executes it will fail. This allows for multiple copies of the same DLL, each with a different version number. Shared components can be managed by the .NET Framework's Global Assembly Cache (GAC). The GAC is not covered by this doc ument.

By allowing developers to be picky about the versioning of any external modules referenced by their assembly they can ensure that their code users DLL's they expect. This can ensure that a programs execute is not tampered with by altering one of it s dependant DLL's. Also this potentially reduces a lot of unexpected errors when a program uses a DLL that has not been tested by the developer.

It is worth noting that strong names themselves do not imply that an assembly can be trusted (or that it is fr om a trustworthy developer). This is because anyone is able to create a strong -named assembly and combine it with any digital certificate. However, a Strong -Named created with the private key from a

digital certificate from a publicly known, and trusted, p rovider can be used as proof of an assembly's origin and ensures that it hasn't been tampered with.

Role-Based Security

The .NET Framework comes with many inbuilt methods of establishing *identity.* The two parts of *role-based security* are authentication an d authorisation. The need for authentication and authorisation is increasingly important in today's applications. Users are given roles and responsibilities. Depending on these factors determines what type of permissions and content are granted.

Authentication is the process of examining credentials and establishing an identity of the principal. Out -of-the-box the .NET Framework provides support for common authentication protocols , including[v]:

- Basic
- Digest
- NTLM
- Kerberos
- SSL/TLS certificates

The .NET Frame work also supports many authentication providers, including [iv]:

- Forms-based (Cookie) Authentication. (Eg. Simple web login page using a database in the back -end).
- Passport Authentication. This is Microsoft's centralised authentication system currently used in *Hotmail*, *MSN Messenger* and other online services.
- Windows Authentication. This allows transparent login and users the username and password from their current Windows session.

In addition the Framework is capable of *impersonation*. This is the process in which a user accesses the resources by using the identity of another user.

Microsoft has given a wide range of choices when it comes to authentication. For those who have tried to pro gram applications using the *Identity* and *Principal* objects they have t ried to make it as easy as possible without limiting the developer any functionality as they would with their own custom authentication modules. In doing so, Microsoft is encouraging developers to use their Role-Based security model as apposed to that or a third party which might not offer the same level of support, or worse still a authentication system written by the developer themselves who has little knowledge on security.

Cryptography

The .NET Framework includes cryptographic functions for encryption , digital signatures, hashing and random number generation [vi]. These are all implemented by well-known and proven algorithms, including [vii]:

- RSA
- DSA
- Rijndael/AES
- Triple DES, DES
- RC2
- MD5
- SHA1, SHA-256, SHA-384, SHA-512

This helps to ensure that programs will us e these well-proven algorithms and not risk comprise by using less known or flawed algorithm.

Isolated Storage

Isolated storage is a real powerful feature for programmers. It allows them to be able to save files to the drive locally without breaking any f ile permissions. This is achieved by the Framework allocating some space on the disk locally. Where this space resides is not known by the program at all. The Framework gives no access to any drive, registry or file information and from the developer's prospective it is just an empty area of space that can be used to store files locally. Each assembly run is allocated its own separate bit of space and therefore t cannot access any files that another assembly may used to store information.

Summary

In summary Microsoft's .NET Framework is an encouraging step forward for both end users and developers.

End users can feel more secure and safe running programs over the Internet and this may lead to a new era of rich-based Internet applications. Though it still remains to be seen how 'safe' they feel about such applications. Also network administrators can roll out all security policies for the Framework from group policy. This should save many hassles and headaches.

Developers are given granular control on the F ramework's security modules and are not limited much by the CLR's strict control of resource access. There is a rich variety of out-of-the-box security and cryptography classes and objects which many programmers should find useful and handy. Role-Based security is targeted at web and ASP developers and, if used correctly, the principal and identity objects are a very powerful, yet flexible, way of authenticating and authorising users.

No other programming platform to date has given developers so much control and granularity over security. Used correctly imperative security can be a very powerful way to extend on the Framework's base security features. Allowing for assemblies responsible for performing sensitive functions to check the credentials of the calling assembly before granting them further access.

With the security feature offered by the .NET Framework, Microsoft is not only 'talking the talk' but they are actually 'walking the walk.' It is certainly a step forward and is also encouraging that Microsoft themselves are setting a benchmark for how software security can be done. Future virtual machines and the like will, no doubt, follow in the direction that Microsoft is attempting to set. However, only time can prove how secure the .NET Framework really proves itself to be.

References

GotDotNET. 'About .NET Security.' http://www.gotdotnet.com/team/clr/about_security.aspx (Oct 2002)

Watkins, Damien. 'An Overview of Security in the .NET Framework.' Microsoft Press 2002.

Clark, Jason. 'Code Access Security and Distribution Features in .NET Enhance Client-Side Apps.' *MDSN Magazine.* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/httpsecurity.asp

Microsoft MSDN Whitepaper. 'Microsoft .NET Framework Security Overview.' http://msdn.microsoft.com/vstudio/techinfo/articles/developerproductivity/frameworksec.asp (Oct 2002)

Bock, Jason. 'Protect Your Code Investment.' *.NET Magazine*. http://www.fawcette.com/dotnetmag/2002_10/magazine/columns/security/ (Oct 2002)

Box, Don. 'Security in .NET.' http://msdn.microsoft.com/msdnmag/issues/02/09/SecurityinNET/print.asp (Oct 2002)

Foundstone Whitepaper. 'Security in the Microsoft .NET Framework.' F*oundstone, Inc and CORE Security Technologies (2002)*. http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/net/evaluate/fsnetsec.asp

Microsoft Whitepaper. 'Security with Microsoft .NET: An Overview. ' http://www.microsoft.com/net/business/security_net.asp

Siddiqui, Mansoor Ahmed. 'Web Services Security in The .NET Framework.' http://www.15seconds.com/issue/020312.htm

End Notes

[i] http://www.microsoft.com/net/basics/

[ii] http://msdn.microsoft.com/netframework/productinfo/overview.asp

[iii] List taken from article: 'Security in the Microsoft .NET Framework', page 7. by *Foundstone Inc*. http://www.foundstone.com/

[iv] List taken from article: 'An Overview of Security in the .NET Framework'. By Dr Demien Watkins.

[v] List taken from article: 'Security with Microsoft .NET: An Overview', page 3. By Microsoft Corporation. http://www.microsoft.com/

[vi] http://www.gotdotnet.com/team/clr/about_security.aspx

[vii] Listen taken from article: 'Microsoft .NET Framework Security Overview', page 8. By Microsoft Corporation. http://www.microsoft.com/