Java and .NET Security

Martin Russold
Antje Melle

Secure Computer Systems
October 2005

Abstract

This report presents some of the aspects concerning security in Java Technology and .NET. It starts with a brief introduction of the Java Technology and .NET. The second chapter focuses on certified code and how it is realized on either side. Afterwards, chapter number three treats the Sandbox model. Finally we give a small summary.

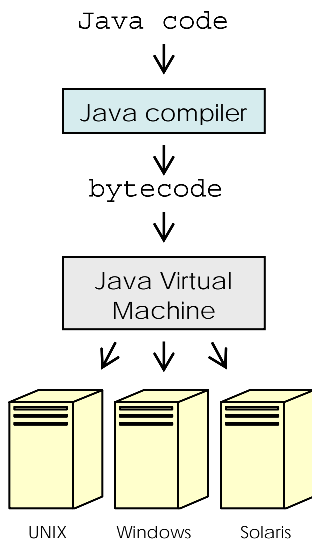
Java Technology and .NET

The following section gives a general review over Java and .NET technologies by comparing both with each other.
The Java Technology supports the programming language with the identical name Java while the .NET Technology allows four different languages: C++, C#, Visual Basic (VB) and JScript.
Java and .NET compilers do not directly generate machine code. Instead after compilation a higher level intermediate code is the result. The advantage of this intermediate code is its machine-independency what eases interoperability. Figure 1 shows the notations used in Java and .NET technology.

Aspect	**Java Technology**	**.Net**
Programming languages	Java	C#, C++, Visual Basic, JScript
Name of the intermediate code	Bytecode	Microsoft intermediate language (MSIL)
Execution environment	Java Virtual Machine (JVM)	Common Language Runtime (CLR)
Predefined classes	Java Application Programming Interface (Java API)	Framework Class Libraries (FCL)
Collection of files	JAR archives	assemblies

Figure 1: Comparison of Java Technology and .Net technology

Java intermediate code runs in every environment with a compliant Java Virtual Machine while .NET intermediate code execution environments are only available for different Windows platforms [1]. The compilation processes are depicted in Figure 2.



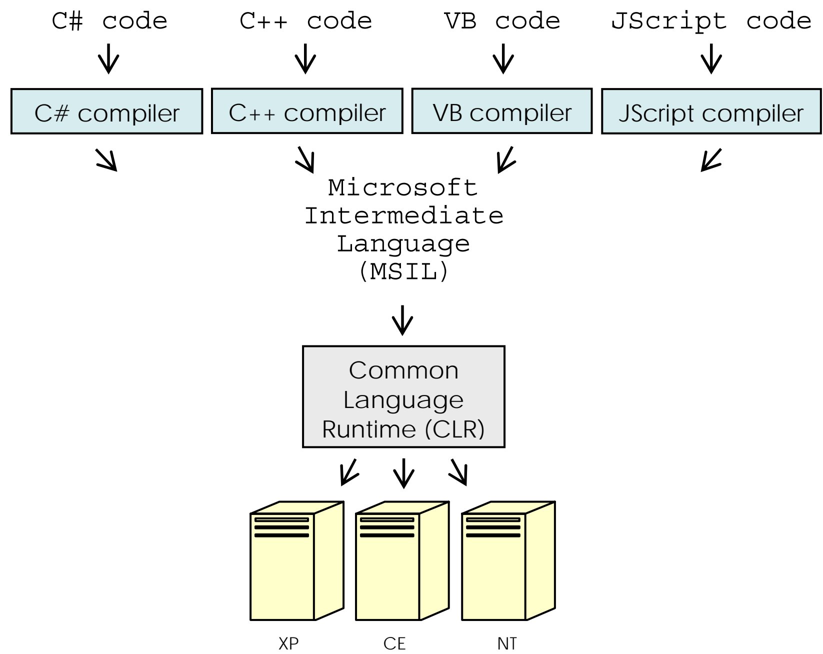


Figure 2a: Java compilation overview

Figure 2b: .NET compilation overview

Both technologies offer a large collection of software components with useful capabilities e.g. for creating user interfaces or communications.

Certified Code

In this chapter we introduce the term certified code. Starting with a brief explanation on how code can be certified in general. Afterwards the realization of certified code in both .NET and Java-Technology will be presented.

In the area of certified code usually the following scenario is given: There are two parties participating, a code publisher (server) and a consumer (client). The server wants to distribute software which the client wants to run on its computer. For security reasons, the client has to be enabled to check properties concerning the security of the software. The questions that have to be answered are: “Who actually wrote the software?” and “Was the software altered by a third party?” Depending on the result of these checks, the client can decide whether the software should be executed or not.

To summarize, certified code is a possibility to inspect the origin of software (authentication). Furthermore it can also verify the integrity of the software. The question “How can code be certified?” remains. Generally this is done by digital signatures. In order to achieve the above given goals of certified code, authentication and integrity, a digital signature has to have the following properties [2]:

- Verifiable: Anyone should be able to validate a signature.
- Unforgeable: Only the original owner of a signature is able to attach its signature to a document.
- Nonreusable: It should be impossible to "lift" a signature off one document and attach it to another.
- Unalterable: It should be impossible for anyone to change the document after it has been signed, without making the signature invalid.
- Nondeniable: It should be impossible for the signer to disavow the signature once it is created.

The following paragraph explains how certified code is realized in .NET.

Certified Code in .NET

.NET supports authentication as well as integrity. Assemblies can be signed to prove publisher identities (authentication). Furthermore assemblies can be signed to prove application identities

(integrity). Both approaches can be used complementary and independently from each other. In order to realize authentication, .NET introduces so-called s*trong names*. The second approach is archived through *publisher certificates*.

A *strong name* consists of the following parts: the assembly's identity, a public key and a digital signature. Thereby an assembly's identity consists of the file name, the version number and optional culture information. The *strong name* is generated from an assembly file using the corresponding private key [3].

Thereby a *strong name* can be viewed as globally unique identifier, with the following properties [4]:

- *Strong names* guarantee name uniqueness by relying on unique key pairs. It is impossible to generate the same assembly name twice, because an assembly generated with one private key has a different name than an assembly generated with another private key.
- *Strong names* protect the version lineage of an assembly. A strong name can ensure that no one can produce a subsequent version of your assembly. Users can be sure that a version of the assembly they are loading comes from the same publisher that created the version the application was built with.
- *Strong names* provide a strong integrity check. Passing the .NET Framework security checks guarantees that the contents of the assembly have not been changed since it was built.

On the other hand, *publisher certificates* establish the authentication of the code distributor (server). Therefore a whole assembly has to be signed with its personal certificate. This signature is afterwards included into the file. This allows the CRL to verify the signature at runtime.

To summarize, *publisher certificates* can be used to give permissions to an application depending on whether we trust the code distributor or not.

Certified Code in Java

Whenever a JAR file is signed, the JAR file itself isn't manipulated. The only difference is that the JAR files manifest is updated and two new files are added to the META-INF directory of the archive. These added files are called: *signature* file and *signature block* file [3].

For each file that is signed, an entry in form of a record is created in the manifest file named MANIFEST.MF. An example is of a manifest file is shown in figure 3

```
Manifest-Version: 1.0
Created-By: 1.3 (Sun Microsystems, Inc)

Name: common/class1.class
MD5-Digest: (base64 representation of MD5 digest)

Name: common/class2.class
MD5-Digest: (base64 representation of MD5 digest)
SHA-Digest: (base64 representation of SHA digest)
```

Figure 3: Example of a manifest file [5]

Each record has the following format: filename and one or more pairs consisting of the digest algorithm and its result. Each individual signer has an own signature file. An example of a signature file is illustrated in figure 4.

```
Signature-Version: 1.0

Name: common/class1.class
MD5-Digest: (base64 representation of MD5 digest)

Name: common/class2.class
MD5-Digest: (base64 representation of MD5 digest)
```

Figure 4: Example of a signature file [5]

The *signature* file contains all digest records of the manifest file at the time of signing. Furthermore the signature file includes an digest for the complete manifest file [3]. This fact implies that a manifest file may not be changed after it was signed. Otherwise a new added file will change the manifest file and invalidate all the existing signatures.
The *signature block* file includes the binary signature of the signature file. Furthermore it contains all the public certificates which are needed for verification and it is always created together with the signature file.
To summarize signing and verifying in Java is actually not based on the physical JAR archive, but on the manifest file. As only the entries of the manifest files are signed. Therefore it gets possible, in opposite to .NET, to add, delete and modify files that belong to the signed JAR archive, as long as these changes do not change the manifest file. So certifying code is more flexible in Java than in .NET [3].

Sandbox

The sandbox is a security mechanism that allows a program that will be hosted on a computer (e.g. an applet) only restricted rights. Therefore the sandbox will protect some of the computers resources.
The sandbox offers the program a limited environment in which it is allowed to "play" and able to use all provided resources, but not able to take any actions beyond the borderlines. The user is able to widen those borders. But to widen the sandbox and allow the program to use more resources necessitates a particular trust to the program.
The idea of the sandbox was first implemented as a Java security mechanism. .NET which was developed a couple of years later adopted and improved this idea.

The Java Sandbox

In pre-internet times computer user purchased shrink-wrapped software. With the internet the area of downloading software was initiated. Therefore it was getting much harder to know which software to trust und which one not to trust. To solve this trouble the idea raised to confine the running environment of a downloaded program. It should be possible to protect resources from potentially evil software. Sun invented the sandbox to realise this request for the Java Technology.

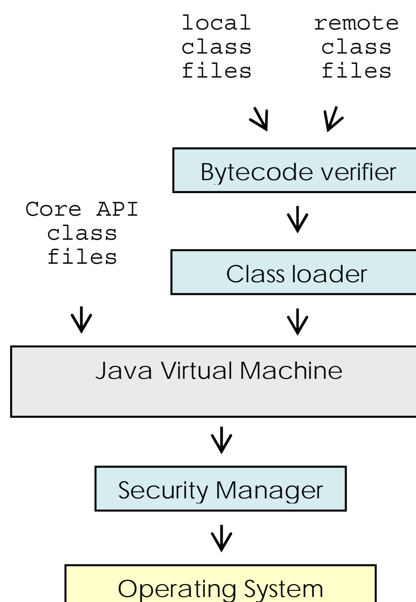


Figure 1: The Java Sandbox (in dependence on [14])

A sandbox can have different levels the user is able to set. The sandbox for untrusted Java applets, for example, prohibits many activities, including [6]:

- Reading or writing to the local disk
- Making a network connection to any host, except the host from which the applet came
- Creating a new process
- Loading a new dynamic library and directly calling a native method

If this security level is chosen it is not possible for the program to execute critical calls and the user is protected from inimical code.

The fundamental components that are used to build the Java sandbox are described in the following. First component is the *bytecode verifier*. Since it is possible that the code is produced by a non-trustworthy Java compiler, the Java run time system doesn't trust the incoming code before it wasn't subjected by the *bytecode verifier* [7]. It proves that Java class files fulfil the Java language rules and doesn't violate access restrictions. Core API class files do not have to be verified by this component.

The *class loader* as second component of the sandbox is responsible for loading the classes into the Java runtime environment. The bootstrap class loader is only able to load the Core API class files [8]. The position of those core classes is specified by the CLASSPATH variable. All other classes have to be searched and loaded from one or more additional class loaders. All classes imported from the network get an extra namespace. This namespace is mapped to a protection domain. A protection domain encapsulates the set of permission. All classes of one protection domain have the same permissions.

Additionally Java has a security manager. This component returns if an operation is permitted during runtime. Therefore the security manager represents the reference monitor of the Java security model [9]. If the operation is not permitted the security manager is able to prevent completion by throwing an exception [10].

.NET and its security architecture

The similar to Java .NET disposes over a class loader. This component is responsible to load classes and to make them available for the Common Language Runtime.

Another component is the just-in-time (JIT) verifier [11]. It is the pendant to Java's bytecode verifier. In addition to the bytecode verifier it checks some metadata that is included in the assembly to seek out if the bytecode is valid [12].

The .Net reference monitor is implemented by two security functions: the *code access security* and the *role based security* [13]. The *role based security* determines the restrictions on resource types and the privileged operation of a user by its identity. When a user authenticates his identity is mapped to a particular role. A role is a administration domain containing users which have to perform similar tasks and need therefore almost the same rights. Dependent on the role of the user, security decision are made.

Code access security determines which resources and operations the code is allowed to perform by its origin. That means that it depends on the codes origin what the code is allowed to do. Thereby a zone concept is used [12] where distinguished between the internet, intranets and the local machine. We can emanate from that code hosted on the local machine is more trustworthy than code downloaded from the internet. Therefore code from the local zone can perform more operation than code from the internet zone. But it is also possible that the level of trust is determined by digital certificates.

Conclusion

Both Java and .Net offer mechanisms to execute untrusted and downloaded code in a secure way. Java was the first environment offering the sandbox for the secure execution. .NET took up the concept and augmented it by the role based and the code access security which allows more granular restrictions. Both environments allow certified code to authenticate the publisher of the software and check its integrity. So the user has the possibility to trust even downloaded code.

References

[1] http://java.oreilly.com/news/farley_0800.html

[2] http://www.securingjava.com/chapter-three/chapter-three-3.html

[3] http://www.onjava.com/pub/a/onjava/2004/01/28/javavsdotnet.html?page=2]

[4] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconstrong-namedassemblies.asp

[5] http://java.sun.com/j2se/1.3/docs/guide/jar/jar.html#Signed%20JAR%20File

[6] http://www.javaworld.com/javaworld/jw-08-1997/jw-08-hood.html

[7] http://java.sun.com/docs/white/langenv/Security.doc3.html#1056

[8] http://java.sun.com/developer/technicalArticles/Networking/classloaders/index.html

[9] Dieter Gollmann, Computer Security, John Wiley & Sons, 1999, ISBN: 0-471-97844-2

[10] http://java.sun.com/j2se/1.3/docs/guide/security/smPortGuide.html

[11] http://www.cs.virginia.edu/~nrp3d/papers/computers_and_security-net-java.pdf

[12] http://www.cgisecurity.com/lib/J2EEandDotNetsecurityByGerMulcahy.pdf

[13] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh06.asp

[14] Scott Oaks, Java Security, O'REILLY, First Edition, ISBN: 1-56592-403-7