SUPERVISING THE SECURITY OF THE MODERN INFORMATICS SYSTEMS

Mihai CARAMAN
Stefan DAN
Daniel BUCATOS
Adrian Virgil CRACIUN

Automatics Department, "Transilvania" University of Brasov, 5 Mihai Viteazu Street, 500174, Brasov, Romania, phone/fax: +40 0268 418836,
mihai.caraman@deurcenteromedia.ro, dan.stefan@unitbv.ro,
daniel_bucatos@yahoo.com, craciun@vega.unitbv.ro

This paper presents security supervising mechanisms for modern informatics systems. We emphasis the .NET Code Access Security technology features and present a new application CAS Scenarios Generator Tool. This tool was designed to deliver source code for various scenarios that are encountered dealing with CAS, to compile and run the generated binaries and to acquire and interpret the results. This paper also presented aspects of the security administration and extensions to the open source network assessment tools.

Keywords: Code Access Security, .NET Framework, network security tools

1. INTRODUCTION

One of the main priorities of any informatics system is to be secure. This issue address from the mobile equipments to personal computers, the corporation networks and not less the internet backbone.

The assurance of security for an informatics system is necessary in order to insure the processing resources and its services, to protect the information confidentiality or to avoid its use as host for new attacks.

The assurance of security can be achieved on two ways: developing secure applications and providing a well-balanced system administration.

The development process for secure applications resides on modern technologies such as: .NET and Java. Those technologies offers new concepts integrated into frameworks dedicated for security.

Security administration includes general techniques such as: application configuration, application maintenance through patches and updates, the usage of antivirus application, firewalls, antispyware as well as tools for vulnerability detection.

This paper approaches the following key aspects:

Security modern technologies: The use of the new .NET security framework. This managed environment drastic diminishes the potential occurrence of buffer overruns and provides code access security to help solve trust, semitrusted and untrusted code dilemma. In this presentation we emphasis the .NET Code Access Security (CAS) features.

The development process of the secure applications: Writing secure code using the .NET security framework and embarrassment of the SD3 strategy: secure by design, by default and in development. We present our new application named CAS Scenarios Generator Tool.

Administration techniques: Setting the CAS security policies.

Open source network assessment tools: Extending and use of Nessus tools for vulnerability detection and attacks prevention.

2. THE SECURITY SUPERVISING SYSTEM

2.1. .NET Code Access Security

2.1.1.1. The Windows Classical Model

The classical security in Microsoft Windows considered only the principal's identity when performing security checks. That means that if the user is trusted the code runs with the person's identity and as a result is trusted and has the same rights as the user.

2.1.1.2. A Definition For Code Access Security

The new .NET common language runtime offered by Microsoft provide managed code that help mitigate some of the security vulnerabilities like buffer overruns and issues associated with fully trusted mobile code like ActiveX.

Code access security allows code to be trusted to varying degrees, depending on where the code originates and on other aspects of the code's identity. Code access security also enforces the varying levels of trust on code, which minimizes the amount of code that must be fully trusted in order to run.

2.1.1.3. Code Access Security Features

Code access security is a mechanism that controls the access code has to protected resources and operations. It performs the following functions:

- Defines permissions and permission sets that represent the right to access various system resources.
- Enables administrators to configure security policy by associating sets of permissions with groups of code (code groups).
- Enables code to request the permissions it requires in order to run, as well as the permissions that would be useful to have, and specifies which permissions the code must never have.

- Grants permissions to each assembly that is loaded, based on the permissions requested by the code and on the operations permitted by security policy.
- Enables code to demand that its callers have specific permissions
- Enables code to demand that its callers possess a digital signature, thus allowing only callers from a particular organization or site to call the protected code.
- Enforces restrictions on code at run time by comparing the granted permissions of every caller on the call stack to the permissions that callers must have.

To determine whether code is authorized to access a resource or perform an operation, the runtime's security system walks the call stack, comparing the granted permissions of each caller to the permission being demanded. If any caller in the call stack does not have the demanded permission, a security exception is thrown and access is refused. The stack walk is designed to prevent luring attacks, in which less-trusted code calls highly trusted code and uses it to perform unauthorized actions.

2.1.1.4. Security Syntax

Code that targets the common language runtime can interact with the security system by requesting permissions, demanding that callers have specified permissions, and overriding certain security settings (given enough privileges). There are two different forms of syntax to programmatically interact with the .NET Framework security system: declarative syntax and imperative syntax.

- Declarative security syntax uses attributes to place security information into the metadata of the code. Attributes can be placed at the assembly, class, or member level, to indicate the type of request, demand, or override we want to use.
- Imperative security syntax issues a security call by creating a new instance of the permission object we want to invoke. We can use imperative syntax to perform demands and overrides, but not requests.

2.1.1.5. Requesting Permissions

Requesting permissions is the way we let the runtime know what our code needs to be allowed to do. We request permissions for an assembly by placing attributes (declarative syntax) in the assembly scope of our code. When the assembly is created, the language compiler stores the requested permissions in the assembly manifest. At load time, the runtime examines the permission requests, and applies security policy rules to determine which permissions to grant to the assembly. Requests only influence the runtime to deny permissions to our code and never influence the runtime to give more permission to our code. The local administration policy always has final control over the maximum permissions our code is granted.

- RequestMinimum specify the permissions that our code must have in order to run.
- RequestOptional specify the permissions that our code can use, but can run effectively without. This request implicitly refuses all other permissions not specifically requested.
- RequestRefuse specify the permissions that we want to ensure will never be granted to our code, even if security policy allows them to be granted.

2.1.1.6. Security Demands

To ensure that only callers that have been granted a specified permission can call our code, we can declaratively or imperatively demand that callers of our code have a specific permission or set of permissions. A demand causes the runtime to perform a security check to enforce restrictions on calling code. During a security check, the runtime walks the call stack, examining the permissions of each caller in the stack and determining whether the permission being demanded has been granted to each caller. If a caller that does not have the demanded permission is found, the security check fails and a SecurityException is thrown. The only demands that do not result in a stack walk are link demands, which check only the immediate caller.

- Demands. We can use the security demand call declaratively or imperatively to specify the permissions that direct or indirect callers must have to access our library.
- Link demands. A link demand causes a security check during just-in-time compilation and only checks the immediate caller of our code. Linking occurs when our code is bound to a type reference, including function pointer references and method calls. If the caller does not have sufficient permission to link to our code, the link is not allowed and a runtime exception is thrown when the code is loaded and run. Link demands can be overridden in classes that inherit from our code.
- Inherit demands. Inheritance demands applied to classes have a different meaning than inheritance demands applied to methods. We can place inheritance demands at the class level to ensure that only code with the specified permission can inherit from our class. Inheritance demands placed on methods require that code have the specified permission to override the method.

2.1.1.7. Overriding Security Checks

Normally, a security check examines every caller in the call stack to ensure that each caller has been granted the specified permission. However, we can override the outcome of security checks by calling Assert, Deny, or PermitOnly on an individual permission object or a permission set object. Depending on which of these methods we call, we can cause the security check to succeed or fail, even though the permissions of all callers on the stack might not have been checked.

- Calling Assert enables the code (and downstream callers) to perform actions that our code has permission to do, but its callers might not have permission to do. A security assertion changes the normal process that the runtime performs during a security check, telling the security system not to check the callers for the asserted permission
- Calling Deny prevents access to the resource specified by the denied permission. If our code calls **Deny** and a downstream caller subsequently demands the denied permission, the security check will fail, even if all callers have permission to access that resource
- Calling PermitOnly has essentially the same effect as calling Deny, but is a different way of specifying the conditions under which the security check should fail. **PermitOnly** says that only the resources we specify can be accessed

2.2. CAS Scenarios Generator Tool

The CAS Scenarios Generator Tool (CAS SGT) was designed to deliver source code for various scenarios that are encountered dealing with CAS. It offers the possibility of choosing the topology and the security enforcements for our application:

- Name the desired assemblies
- Sign the assemblies
- Assign standard and custom security permission attributes per assembly. Select the declarative or interactive syntax.
- For each assembly define methods choosing from a predefined list of functions. This functions specify the system resources that they access: files, environment variables, registry, sockets etc
- Select the call flow between methods
- Assign standard and custom security permission attributes per method: FileIOPermission, SocketPermission, IsolatedStoragePermission, etc
- Assign CodeAccesPermission calls per method: Demand, Assert, Deny

The CAS SGT it is also capable to compile and run the generated binaries acquiring and interpreting the results. It presents the list of possible conflicts and exceptions generated by the lack of right permissions and present hints to avoid dangerous code.

This tool is useful for two main reasons. First we have a base to test our concepts combining different permission requirements and see the possible results. Second we have the source code that can be used for didactical purposes or directly reused in production solutions.

The CAS SG code generation is accomplished by the use of the CodeDom support offered by the .NET Framework. The CodeDom mechanism enables

developers of programs that emit source code to generate source code in multiple programming languages at run time, based on a single model that represents the code to render.

The CAS SGT has an extensible architecture offering an interface for adding new plug-ins. The plug-ins describes in an xml format or directly in a .NET source code file a list of functions that generally access system resources.

2.3. The Code Access Security Policy System

The CAS system is driven by a persisted security policy, offering different configuration containments (called policy levels) for enterprise-wide security configuration and machine-wide and per-user security settings. Security policy can be set using a GUI tool (the .NET Framework Configuration tool), a command line tool for batch scripting security changes, or by programming to the security APIs directly.

2.4. Open Source Network Assessment Tools

Nessus is a free and open source vulnerability scanner. It exposes the Nessus Attack Scripting Language (NASL) specifically designed for developers in order to write their own vulnerability checks. It presents predefined function specially designed to perform network vulnerability tests. NASL language is more portable and architecture independent than C programming language

Supposing that we have implemented a proprietary network protocol on top of a TCP socket server, a plug-in implementation would be to test the input validation.

3. CONCLUSIONS

In this paper we presented security supervising mechanisms for modern informatics systems. We reviewed the .NET Code Access Security features such Requesting Permission, Security Demands and Overriding Security Checks.

We presented the CAS Scenarios Generator Tool, designed to deliver source code for various scenarios that are encountered dealing with CAS, to compile and run the generated binaries and to acquire and interpret the results. We enumerated other security mechanisms such administration of the policy system and open source network assessment tools

4. REFERENCES

[1] M. Howard, D. LeBlanc*: Writing Secure Code*, Microsoft Press, 2003

[2] B. LaMacchia, S. Lange, M. Lyons, R. Martin, K. Price*: .NET Framework Security*, Addison Wesley, 2002

[3] P. Thorsteinson, G. Gnana Ganesh: *.NET Security and Cryptography,* Prentice Hall, 2003

[4] J. Clarke, N. Dhanjani: *Network Security Tools,* O'Reilly, 2005

[5] *Visual Studio .NET 2005 MSDN Library,* Microsoft, 2005