

"Microsoft must deliver 'secure environments' not tools to write 'secure code'" : draft article

Source:

<http://www.derkeiler.com/Newsgroups/microsoft.public.dotnet.framework.aspnet.security/2003-10/0328.html>

From: Dinis Cruz (*dinis_at_ddplus.net*)

Date: 10/30/03

Date: 29 Oct 2003 16:07:02 -0800

Hello

Please see bellow the final draft of an article soon to be published.

I would appreciate your comments and corrections of anything that I might have got wrong.

Best regards

Dinis Cruz
.NET Security Consultant
DDPlus (www.ddplus.net)

Microsoft must deliver 'secure environments' not tools to write 'secure code'.

The latest Microsoft development environment, the .Net Framework (currently on the 1.14 version) contains a new security paradigm called Code Access Security (CAS).

The Code Access Security (CAS) main concept is that code should be executed based on its origin, its signature and its function. As an example, when a 'scientific calendar' (from an external software supplier) is executed by the system's administrator on a server, it should not run with the logged in user security privileges (i.e. administrator rights) and should only have limited access to core windows functions.

Compared with previous technologies, Code Access Security (CAS) is a major step in the right direction, but the problem is that Microsoft still has the wrong approach to security.

Microsoft is focused in providing (i.e. selling) powerful development

environments (such as the windows 2003 API, the .Net framework, the WMI) and powerful (web or windows based) application development tools (such as the Visual Studio .Net/2003 or the Office 2003 development toolkit).

On the security front, Microsoft is focused in providing tools that allow developers to write secure code. The problem is that (today) these techniques (for example CAS) don't really work in the 'real world'.

They are either too complicated or just not functional. For example, with the current 1.14 version of the .Net framework the only way to create secure windows or web application (using the .Net framework in a 'Partially trusted' environment) is to:

- a) Do it within the huge limitations of an 'partially trusted' .Net environment (i.e. most of the .Net functionality is disabled), or
- b) Spend an extra 50% to 100% development time on planning, coding and testing the application so that it can be deployed in 'partially trusted' environments.

With these limitations (in the 'real world') what almost everybody is doing (web or windows developers) is writing applications designed for (i.e. will only work in) 'Full trust' environments.

'Full Trust' environments will allow the application's code full access to the server's Win32 API and .Net core functions (Public or Private). This is almost the same as running with system administrator privileges (even if the application is executed under the security rights of a 'low privilege' account).

And why are today's developers writing such insecure applications? In my view there are two main reasons:

- Reason number 1: "Because they can get away with it!"
- Reason number 2: "Because they have no secure environment to test their applications."

1) "Because they can get away with it!"

In both web and windows application cases, the developers know that their code will be executed in 'Full Trust' environments:

– In the current version of the .Net Framework (version 1.14) all code executed from the local machine is given 'Full Trust'. This means that if the user copies the application to a local folder and executes it, the application will run in a 'full trust' environment.

– Most ISPs providing asp.net shared hosting environments don't change the default IIS 5.0 or IIS 6.0 trust settings ('Full Trust') and force

their client's websites to execute in 'partially trusted' environments.

Even Microsoft's own applications don't run in 'Partially trusted' environments! (Microsoft's developers also know that their applications will run with 'Full Trust'). Note that in the new office 2003, the only way to execute .Net assemblies (now used instead of VBA) is to run them from 'known' locations with 'full trust'.

This all comes down to the decade's old security concept: 'Code should be created to execute with the minimum required permissions and be given access to the minimum required Operating systems functions', or in another words 'don't run and install programs as administrator'.

At least now (with .Net and CAS) it is theoretically possible to write 'secure applications' for 'secure environments', previously it was almost impossible.

2) "Because they have no secure environment to test their applications."

– How can one write secure code if one doesn't have a secure environment to test it?

– How can one write secure code for secure environment if there is almost no documentation about how to do it?

Just like the ZX spectrum programmers managed to create amazing games in 48k or 128k, the .Net developers (if they are "forced to") will be able to create amazing applications within the limitations of an secure environment.

So, my solution is for Microsoft to release secure environments which make sense and work!

The Microsoft's 'Secure by Default' although is a step on the right direction, it is still very far from what it should be. The main difference between windows 2000 and window 2003 is that the later has more 'options' disabled by default. But since to make the server do what we want it to do, we have to enable these functions; the 'secure by default' in most cases is only a cosmetic operation.

What we need (as system operators) is to be able to create windows or hosting environments that are very secure by customizing it to the requirements of the business applications.

This cannot be done in an environment where Microsoft is producing a new Operating systems every three years and new applications every year! The reason this is a major problem is that Microsoft's most senior and experienced developers, will be focused on the 'next' versions, instead of creation 'real live' solutions for the 'current'

or 'previous' versions of their Operating Systems.

Another problem with this 'quick releases cycles', is that Microsoft successfully manages

To avoid any direct consequence of producing and selling software that contains security vulnerabilities, because:

- a) By the time serious security vulnerabilities are discovered by the community (several security incidents later) and that version of Microsoft's operating system is considered 'insecure', Microsoft will release a new version
- b) The new version provides the solution of 'upgrade and the security problems will be resolved'
- c) Since new versions of Operating Systems always take a couple of years to be widely used, we are back in point a)

As long as Microsoft manages to produce security patches for the 'known' security vulnerabilities, they will never be Microsoft's responsibility.

The official reason why the clients were affected by an exploit of security vulnerability, is because they didn't apply the latest patch (that could have crashed the server) or because they didn't securely configure the server.

I'm being ironic, but even Microsoft now recognizes that their software updating system is not a) practical, b) its installation can have serious side effects and c) sometimes it doesn't solve completely the reported vulnerabilities.

If you look at the worms and viruses that we had this year (2003), think what would have happened if the malicious code had formatted the hard drives after successfully infecting 10 servers or workstations!

This (or worse) will happen in the future unless these worms/virus when infecting a machine only have limited access to it (not like now, that they have full administrative access, or run in 'Full Trust' environments).

If Microsoft spent as much money, energy and focus in creating secure versions of windows 2000 and 2003 servers and windows 2000 and XP professional workstations, then they would be able to:

- 1) Develop tools for system administrators that would help them to test their security set-up, and help them to securely configure their Operating Systems / Business Applications (think the version 5.0 of the Baseline Security Analyser, the current version is the 1.1)
- 2) Provide developers with documentation on 'how to write for secure (or 'Partially Trusted') Environments. I don't mean general examples or explanation on how the technology works. What is required are

dozens or hundreds of specific examples of 'real live' projects that were successfully designed, developed and deployed in 'partially trusted' environments.

3) Create software (for example: applications and windows services) that can be installed and executed in 'Partially Trusted' environments in the 2000/2003/XP operating systems

4) Create a 'Partially trusted' brand, or maybe a 'Trustworthy application' brand. In this scenario the clients would be able to test the security of the new application and put pressure on their software suppliers to deliver secure products (i.e. develop applications that can be installed in 'Partially trusted' environments)

Why should WinZip and Acrobat Reader have full access to our systems?

What I am defending in this article makes sense from a security point of view, but not from Microsoft's short term commercial point of view (in the long term, Microsoft would benefit enormously from this).

So, until there is enough pressure from Microsoft's development community and their clients saying 'more security not more features', Microsoft will continue to have their 'trustworthy computing' initiative going in the wrong direction. And the security outbreaks that we have today will not stop anytime soon.

Dinis Cruz is a Security Consultant currently working for several UK governmental Departments, International Corporations and ISPs. He is a specialist in .Net Security and is the creator and main developer of the Open Source web application ANSA (Asp.Net Security Consultant). He is also the managing director of DDPlus; a UK based IT Security Company (www.ddplus.net). Dinis can be contacted on dinis@ddplus.net
