

asp.net vulnerability

Source:

<http://www.tech-archive.net/Archive/SharePoint/microsoft.public.sharepoint.portalserver/2004-10/0590.html>

From: Tim Philomeno (*tphilomeno_at_sparling.com*)

Date: 10/11/04

Date: Mon, 11 Oct 2004 11:20:53 -0700

Does anyone know if the fixes that are being suggested will affect how Sharepoint works?

<http://www.microsoft.com/security/incident/aspnet.msp>

or

----- cut from my

email -----

-----Original Message-----

From: Windows NTBugtraq Mailing List

[mailto:NTBUGTRAQ@LISTSERV.NTBUGTRAQ.COM] On Behalf Of Mark Burnett

Sent: Thursday, October 07, 2004 8:12 PM

To: NTBUGTRAQ@LISTSERV.NTBUGTRAQ.COM

Subject: More details on ASP.NET vulnerability

There has been some confusion with the ASP.NET forms authentication issue and I wanted to clarify some points. First of all, this is really an authorization issue, not an authentication issue. This sounds trivial but the difference helps to understand what's happening here. Authorization is what determines if authentication needs to happen.

Normally when you make a request for a protected resource, ASP.NET checks the web.config to see if there is an authorization rule for that resource. If there is no match, it checks the authorization section in the web.config of each parent application all the way up to machine.config, which by default allows everyone to access everything.

The problem here is that by using a backslash, the code that compares the path string and the protected resource always fails. It does not properly match the path string to anything in web.config and eventually ends up in the machine.config, which allows access (note that this current vulnerability applies to the backslash, but it could potentially be any form

of obfuscation that IIS might allow). Since it does not find any rules requiring authentication, it allows access to the resource without prompting the user for credentials, because it sees no need to do so. Therefore, it is not an authentication issue because it never gets to that point. Also, this means that it does affect both Forms and Windows authentication (assuming the NTFS permissions allow access to the ASP.NET process).

One problem in particular with Windows authentication is that ASP.NET will not perform any access checks if there is no valid WindowsIdentity. Therefore (without going into a long explanation of how this works) Windows authentication using file authorization is also vulnerable.

Here are some ways to prevent this and other related attacks that might appear in the future:

1. Follow the best practice of least privilege and change your machine.config to deny all users and all verbs by default. This requires that you specifically allow access to any unprotected resources (especially the login page). One problem here is that ASP.NET does not allow wildcards in the location elements, so you must explicitly allow access to each directory individually, which might be a lot of work on some sites. If changing machine.config is not practical, make sure that each web.config for protected content areas has a default rule that denies access.
2. Use code-based authorization to explicitly check authorization within each module. This is a lot of work but an excellent secure coding practice. Note that any requests exploiting this vulnerability will have an identity object with empty strings so all code-based authorization should fail.
3. Use URLScan, even on IIS 6 servers.
4. On IIS 6, set EnableNotUTF8 and PercentUAllowed to 0 (HKLM/CurrentControlSet/Services/HTTP/Parameters). This does not affect this issue directly but limits exposure to future encoding attacks.
5. Use Microsoft's global.asax code (KB 887459) but also consider expanding environment strings, checking for DOS device names, using a regex to check for known invalid characters, and then finally checking to see that the path is still within the web content directories. You might also want to check the user principal here and perform code-based authorization checks. You may find that doing all this considerably affects performance so you would have to customize this for your particular web site.
6. Use Microsoft's HTTP Module (<http://support.microsoft.com/?kbid=887289>), which is essentially the same code as the global.asax example, but it applies to all sites on the server and is quick and easy to install.

This problem could have been avoided in two ways:

First, and most obviously, thoroughly following the best practice of filtering application input. Microsoft provides a simple fix at

<http://support.microsoft.com/?kbid=887459>. Such a simple vulnerability really was foreseeable and the check should have been in the original product.

Second, following the best practice of least privilege, which results in a system that fails closed rather than failing open. Having the `<allow users="*" />` entry in `machine.config` is equivalent to having a default firewall rule that allows everything that doesn't match any other rule. Although I understand there are likely other issues involved in their decision to do this, the most secure method is to deny by default allowing authentication to fail closed.

Mark Burnett

Hacking the Code: ASP.NET Web Application Security

<http://www.hackingthecode.com>