Database Security Brief
David Litchfield
18th November 2005

Q: Why should I never logon to a Windows database server if I've got admin privileges?

A: It is generally accepted as good security practice that database servers, or any server for that matter, should be configured to run in the security context of a low privileged user. This way, if the server is vulnerable to an arbitrary code execution flaw which is then subsequently exploited by an attacker, any code will run with this user's privileges thus helping to mitigate the risk and protect the system. Indeed, it doesn't even have to be a flaw: most database servers have some kind of functionality where arbitrary code can be introduced; for example an extended stored procedure. All the effort of configuring the server to run as a low privileged user could be for naught though if a Windows administrator ever connects to the server. Let's examine the reason why.

Most database servers when using operating system authentication either use LogonUser() function or the AcceptSecurityContext() function. When the user is successfully authenticated both of these functions create a security token. This security token is then used for authorization purposes. A security token can be thought of as a bunch of keys and an administrator's token has the full set - they can get into every "door" on the system. There are different kinds of tokens such as a primary token and an impersonation token. Calling LogonUser() with the LOGON32_LOGON_NETWORK flag, as a network server would do, creates an impersonation token as does AcceptSecurityContext(). Whilst an authenticated user remains connected to the database server their token remains in the server process' address space. If that database server is vulnerable to an arbitrary code flaw then an attacker could launch a Sleeper exploit. A Sleeper exploit does exactly what it sounds like it does - it sleeps until the conditions are right before it delivers its final payload. In this case the Sleeper exploit would awaken, scan memory for security tokens, either by looping from 0 to 0xFFFFand testing handles to see if they're tokens or by detouring the relevant logon functions, and test them to see if they're admin tokens. If not it sleeps again; but if one of the tokens belongs to an administrator then the exploit snags it. Going back briefly to our discussion on tokens, a primary token can be used to create processes as another user whereas an impersonation token cannot. However it is possible to create a primary token from an impersonation token using the DuplicateTokenEx() function. Once the exploit has snagged the administrator's impersonation token it would then duplicate it to create a primary token. Once armed with a primary token the exploit can then create new processes on the database server with administrator privileges.

As can be seen, even if you run your database server as a low privileged user it is still possible for an attacker to gain administrator privileges if you allow administrators to connect to the server. It should be a matter of security policy that no-one is allowed to connect to a database server or any network based server that requires OS authentication with Windows administrator privileges. In the absence of any kind of technological solution this is the best way to help mitigate the risk.

References:

DuplicateTokenEx:
http://msdn.microsoft.com/library/en-us/secauthz/security/duplicatetokenex.asp
AcceptSecurityContext:
http://msdn.microsoft.com/library/en-us/secauthn/security/acceptsecuritycontext__ntlm_.asp
LogonUser:
http://msdn.microsoft.com/library/en-us/secauthn/security/logonuser.asp