**Remote Debugging**

Published by [marco](http://earthli.com/news/view_user.php?name=marco) on 20. May 2009 05:07:24

When a .NET application exhibits behavior on a remote server that cannot be reproduced locally, you’ll need to debug application directly on the server. The following article includes specific instructions for debugging ASP.NET applications, but applies just as well to standalone executables.

### Prerequisites

There are several prerequisites for remote debugging; don’t even bother trying until you have all of the items on the following list squared away or the Remote Debugger will just chortle at your naiveté.

* The *SERVER* must have the *Visual Studio Remote Debugging Monitor* installed.
* The firewall must be opened for *Visual Studio* on the client (which means that ReSharper sees other instances); remote debugging involves two-way communication.
* A local user, *BOB*, with administrator rights on the client machine.
* A server user, *BOB*, with administrator rights on the *SERVER* machine.
* The names must match.
* The monitor must be started on the *SERVER* using *BOB* (using “Run as…”)
* If you’re not debugging in the same domain, then you have to change the options to use the *Server name* in the options to “*BOB@SERVER*”

Before you think you can get all fancy and simply debug remotely without authentication, know this: unauthenticated, native debugging does not support breakpoints, so forget it. You’ll technically be able to connect to a running application but, without breakpoints, you’ll only be able to watch any pre-existing debug output appear on the console, if that.

### Firewall ports

The following ports must be open in order for Remote Debugging to function correctly in all situations:

**Protocol** **Port** **Service Name**

TCP 139 File and Printer Sharing

TCP 445 File and Printer Sharing

UDP 137 File and Printer Sharing

UDP 138 File and Printer Sharing

UDP 4500 IPsec (IKE NAT-T)

UDP 500 IPsec (IKE)

TCP 135 RPC Endpoint Mapper and DCOM infrastructure

Additionally, the application “Microsoft Visual Studio 2008” must be in the exceptions list on the client and “Visual Studio Remote Debugging Monitor” must be in the exceptions list on the server.

### Recommendations

Once you’ve satisfied the requirements above, you should probably also heed the following tips: it’s best to read about them now rather than learn about them the hard way later:

* Make sure to turn off recycling and auto-shutdown for the AppPool while debugging, so you don’t run the risk of your PID suddenly being gone.
* Make sure that you’re using debug versions of all assemblies where you want to debug or you’ll be staring at IL assembly code more often than you’d like.
* Make sure your local source code is in-sync with the source code on the *SERVER* or you’ll be debugging on the wrong lines at best or be staring at IL assembly at worst.
* It’s best if the path to your local symbols is also valid and writable on the server so that symbols cached during remote debugging can be stored on the server. Check the “Options..Debugging..Symbols” to change that path if you need to. (there’s more about this below if this doesn’t make sense)

### Test Run

Here are steps you can follow to debug an application remotely. These steps worked for me, but the remote debugging situation seems to be extremely hit-or-miss, so your mileage may vary.

1. Open your web project in Visual Studio and compile it in debug mode.
2. Outside of Visual Studio, build a deployment version of the web site and copy it to the *SERVER*.
3. If this is the first time setting it up, move the application to its own *ApplicationPool*, so you can detect it more easily later.
4. If you haven’t already, install the *Visual Studio Remote Debugging Monitor* to the *SERVER*.
5. Make sure you have a local user on that machine with your own user name, *USER*.
6. Start the *Visual Studio Remote Debugging Monitor* using “Run as…” and entering *USER* on that server.
7. When it has started, select “File..Options” from the menu and change the server name to *USER@SERVERNAME*.
8. From within Visual Studio, select “Debug..Attach to Process” from the menu.
9. In the dialog, specify the *USER@SERVERNAME* you used in the debugging monitor above and hit *Refresh*.
10. Scroll down until you see the “w3wp.exe” processes

You’ve set up the server and attached to it so far. If anything has gone wrong, check the troubleshooting section below to see if your problem is addressed there. Now, the next steps are optional if you think you can identify your process without knowing the PID (Process ID). This is generally the case only when yours is the only .NET application deployed to that server. In that case, your process is the “w3wp.exe” process which includes “managed code”. If you don’t know your PID, follow the optional instructions below to figure out which one is yours.

1. From the client machine, download the attached script “[ASP.NET PID Detector](http://data.earthli.com/news/attachments/entry/2168/asp.net_pid_detector.zip)” and open it in a text editor.
2. Change the *machineName*, *appPoolName* and *url* to match the settings for your application on the *SERVER*. (this is the reason we put our application into its own application pool at the beginning.)
3. Save the file as a different name (probably with the machine name and server in the title).
4. Execute the file and follow instructions; it will probably launch your web site in IE. It will probably also claim to have failed. Run it again and it will give you the PID of your application on the server.

If that didn’t work, then you probably aren’t configured to query WMI remotely; your only options are to try to run it remotely using the instructions and tips below or to run it from the server.

* If you have remote desktop access to the server, then copy the script to the server and configure the batch file to query the local script and server (recommended).
* Turn off the Windows Firewall on the server completely (not recommended if the server is open to the internet).
* Follow instructions at [Enable WMI (Windows Management Instrumentation)](http://www.poweradmin.com/help/enableWMI.aspx) to enable remote administration through the firewall. Not only must you execute a special command to configure the Firewall (only available from the command line) but your user must also have the correct permissions (also not recommended, as enabling WMI can open up the server in unexpected ways if you don’t know what you’re doing). I did not attempt this, as I could simply run the PID-detector from the server.

Once you have the PID in hand, continue:

1. Select the “w3wp.exe” process with your PID and double-click it to attach to that process.
2. It will ask whether remote symbols can be stored on the server in the given location. You should say yes but it will try to save those symbols on the server using the *same path you use for storing symbols locally on your development machine*.[[1]](http://earthli.com/news/view_article.php?id=2168#footnote_DRAFTABLE_ENTRY_2168_1_body)
3. Set a breakpoint where desired; the breakpoint should be solid red. If it is, you’re done.
4. Browse the application in IE to trigger the breakpoint and debug away.

[1] I’m honestly not sure whether this is required or not, but I allowed it and it worked. It may also work without caching the symbols if the path can’t be written.[↩](http://earthli.com/news/view_article.php?id=2168#footnote_DRAFTABLE_ENTRY_2168_1_ref)

### Troubleshooting

As you can probably tell from the massive list of prerequisites and recommendations as well as the 20-step guide to triggering a breakpoint, there’s a lot that can go wrong with Remote Debugging. It’s not insurmountable, but it’s not something you’re going to want to attempt unless your job pretty much depends on it. These are some of the errors I encountered along the way and how I addressed them.

Unable to connect to the Microsoft Visual Studio Remote Debugging Monitor named ‘USER@SERVER’. The Visual Studio Remote Debugger on the target computer cannot connect back to this computer. Authentication failed. Please see Help for assistance.

You need to create a local administrator with the same password as the one you’re using on the server to run the debugging monitor.

Unable to connect to the Microsoft Visual Studio Remote Debugging Monitor named ‘USER@SERVER’. The Visual Studio Remote Debugger on the target computer cannot connect back to this computer. A firewall may be preventing communication via DCOM to the local computer. Please see Help for assistance.

You opened the firewall, but only for computers on the same subnet. The computer to which you are connecting is probably not on the same subnet, so you’ll need to go to the firewall settings and open them up all the way (Visual Studio will not ask again). To edit the firewall settings, do the following:

1. Open the “Windows Firewall” control panel.
2. Select the “Exceptions” tag.
3. Scroll to the “Microsoft Visual Studio 2008” entry and double-click it.
4. From the dialog, press the “Change Scope” button.
5. Change it to “Any computer (including those on the Internet)”.
6. Press “Ok” three times to save changes.

It’s also possible that the Remote Debugger is being blocked on the server side. To address this, run the “Visual Studio 2008 Remote Debugger Configuration Wizard” again; if the wizard wants to adjust firewall settings, let it do so (for internal or external networks, as appropriate to your situation – if you’re not sure, use external). To make sure that the settings were applied, run the wizard again; it should ask you about running the service, but should no longer complain about the firewall.

If it still complains about the firewall, then you’ve got another problem, which is that the setup is having trouble adjusting the settings for the firewall but isn’t telling you that it’s utterly failing when it attempts to do so. Verify that you’re running the wizard as a user that has permission to adjust the firewall settings.

Unable to connect to the Microsoft Visual Studio Remote Debugging Monitor named ‘USER@SERVER’. Logon failure: unknown user name or bad password. See help for more information.

The user with which you are executing Visual Studio on the client does not exist on the server or has a different password. In order to avoid adding useless user accounts to the server’s domain, you should restart your IDE using “Run as…” to set the security context to the same user as you have on the server.

You can impersonate other users, but you have set a registry key; see [Remote Debugging Under Another User Account](http://msdn.microsoft.com/en-us/library/aa291278%28VS.71%29.aspx) for more information. This doesn’t help though, if the user you are trying to use doesn’t even have an account on the remote machine.

### Conclusion

Remote debugging sounds way cool and is the major difference between the Standard and Professional versions of Visual Studio, but it’s not for the faint of heart or the inexperienced. If you Google around a bit, you’ll notice that most people get a big heap of epic fail when they try it and I’ve tried to make as comprehensive guide to remote debugging as my own experience and time constraints allowed.

Here’s hoping you never have to do remote debugging (write a test instead! \*smile\*) but, if you do, I wish you the best of luck.