CFHTTP, IIS 8 and Server Name Indication (SNI)

Posted At : July 22, 2013 12:35 PM | Posted By : Mark Kruger
Related Categories: Coldfusion Troubleshooting

Guest Post by *Wil Genovese*

(Muse Introduction)
Most readers know that the Muse is deeply indebted to a large and talented group of developers working here at CF Webtools. These folks solve problems and undertake Herculean programming tasks on a daily basis. They are constantly making me look good and I would not be able to play golf or spend the day wise-cracking in IM and tormenting my assistant Melissa without them on my side. Among these folks is one of my favorite characters, CF guru Wil Genovese. Wil has worked with us for a few years now and he writes an excellent blog at **Trunkful.com**. If you have not already done so, you should add it to your list of must read blogs.

Meanwhile, a few days ago Wil was trying to troubleshoot a head scratching issue with CFHTTTP and SSL. Now such issues almost always come down to getting the certificates properly installed in the keystore, using the correct URL (correct in all respects for the certificate), name resolution and SSL protocol levels (as in "do you need to lower Java's draconian SSL defaults to allow for less secure protocol"). After beating his head against the wall repeatedly Wil finally decided the issue was on the other end - the certificate on the server was somehow wrong, misconfigured or behaving unexpectedly. I thought this was dubious at best, but as is so often is the case the Muse was wrong and Wil found out (with apologies to Monty Python) something completely different. It turns out a new feature in IIS 8 (Windows Server 2012) was the culprit. Since this setting affects all Java versions prior to 1.7 and *even affects CF 10* ***on*** *Java 1.7*, you should probably pay attention. My guess is that you will run into this issue eventually - given the ubiquity of IIS and the coming upgrades to Windows server 2012.

Anyway, I invited Wil to write the following entry detailing his findings. If you want to know more read on:

(Wil writes)
Here's the scoop. We were working on setting up payment processing using ColdFusion 8.0.1 (yes, I know it's ancient but the client is planning to upgrade to CF10 soonish) and it needed to communicate with a client's .NET server via secure CFHTTP (meaning over SSL). The problem was that SSL communications were failing. The error (below) was *I/O Exception: peer not authenticated.*

For several hours we tried everything from importing the SSL certificate into the keystore to creating a separate keystore and including it in the jvm.config arguments. We checked name resolution and tried different Java versions. The issue persisted even after upgrading the JVM to 1.6.0_45. We even tested from ColdFusion 9.0.1 and ColdFusion 10u9 running on Java 1.6.0_29 and nothing was working. Usually we can resolve SSL issues in short order. This issue, however, was beginning to seem like something on the targeted .NET server was preventing SSL communications - except for

one nagging fact. When using a web browser *on the CF server* we could access the payment gateway web service url via SSL with no problem. So SSL was working and all tests indicated that the SSL certificate was installed correctly. What could be the problem?

After discussing the issue a bit more with them and explaining our dilemma the customer decided to grant us access to their payment gateway .NET server. As soon as I logged on I began to get some clues. The server was running Windows Server 2012 and IIS 8. Both are new to me, but with the dogged determination that is the hallmark of a CF Webtool's developer (actually the Muse added that last part... but I was definitely determined) I set out to learn something new. After getting used to the new UI I found a setting for SSL in IIS that I've never seen before; *Use Server Name Indication (SNI)*. What the heck is SNI? In addition, instead of the IIS site having the SSL Cert specifically assigned to it, the setting was set to use the *Centralized Certificate Store*. Again what is this?

(Host names are redacted)

A few Google searches later I learned that *Server Name Indication* (SNI) is the HTTPS equivalent of Name Virtual Hosts for HTTP. Wow! Remember back in the day (well the Muse and I do anyway) when there was a one to one relationship between IP addresses and websites. Web servers couldn't serve more than one web site from a single IP. But the addition of the "host header" setting to the HTTP protocol allowed the web server to "figure out" what site to use to serve up content - allowing for fewer IP addresses to be needed for the grand World Wide Web experiment. SNI does the same thing for SSL certificates. Up until now you needed a one to one relationship between IP addresses and certificates - one IP to one standard certificate (we'll leave wild cards for another post). But now, SSL will no longer need a dedicated IP address per certificate. So multiple certificates combined with multiple host headers can serve secure content from a single IP address. If you want to dig deeper (and I'm still digging myself) here's the wiki page for **Server Name Indication**.

Granted the SNI protocol has been a part of TLS for a while, but it has only become practical recently. Indeed, it's only in IIS 8 that Microsoft made it available. Leveraging SNI, MS also created Centralized Certificate Store - a neat new feature that allows you to create a central location for Certificates that all of the servers in a cluster can access. Thus, implementing or updating a certificate for a groups of servers is (theoretically) a simple process of updating a single Centralized Certificate Store. I won't go into the details of SNI and CCS - mainly because I am still learning them myself - but let's talk about how this affects ColdFusion and CFHTTP over SSL.

(Muse writes)
Before Wil goes further I wanted to interject a thought on how TLS (SSL) handshakes work. On the client side - a browser or some agent like CFHTTP - TLS sends a request to an IP address. As part of the handshake the server responds with 2 items - a certificate and the name of the server (the FQDN as in www.example.com, that goes with the cert). The client then examines the *name* and determines if it matches the desired URL it is asking for. If it *does not* match then the client can do a variety of things. It can warn you for example. Ever get that little message about "some things on the page being insecure" or the message telling you the certificate is bogus and asking if you want to proceed? That's often TLS (SSL) doing it's job matching the domain string to the cert that has been proffered and (of course) verifying the certificate itself.

But it's also why a one to one relationship between IP and certificate has always existed. Think about it carefully - in order to match the certificate the *server* provides the certificate and name to the *client* and the client determines what to do with it. The server has no access to the "host header". That host header is buried in the HTTP request and the server can't "see" that information until *after* the handshake. All the server knows is that it's getting an SSL request to an IP and it has a cert bound to that IP. What has to change in order for the server to pick and choose from among more than one certificate? It has to know *in advance* what domain string (what FQDN) the client is requesting. And that's exactly the problem the SNI extension solves. SNI requires that the server string be sent along with the initial request for an SSL connection. Instead of saying "give me SSL for IP 1.2.3.4" the client now says *"give me SSL for www.example.com"*. Now let's let Wil finish shall we?

(Wil writes)
One of the gotchas is that the client/browser needs to be able to use SNI. It's an extension of the TLS protocol so the client needs to be able to be able to use TLS. Most but not all clients/browsers are compatible. While Java can use TLS (also sometimes referenced as SSL 3.0 - see this **Muse Post** for a good explanation of TLS), it's only since Java 1.7 that Java can use TLS with the SNI extension. This gotcha prevents ColdFusion, while running on Java 1.6 and older, from working with SSL via CFHTTP when the server being targeted is configured for SNI. In fact ColdFusion 8 will *never* work with SNI as it can only run on Java 1.6 and older. ColdFusion 9 and ColdFusion 10 at least have a chance because both are capable of running on Java 1.7.

To follow through and give complete info for this blog post I upgraded a couple of my dev work stations for further testing. Flash forward and after doing multiple updates and testing there is still no joy. After updating ColdFusion 10 to update 11 and configuring it to run on Java 1.7.0_25 I gave the code another try. I re-enabled SNI on the target IIS 8 installation. As I feared ColdFusion 10 could not communicate with the remote server over SSL. Disabling SNI allowed CFHTTP to work again. There is probably no point in testing a CF9 server though we do have a couple CF9 servers running on Java 1.7 at CF Webtools.

In my view this could be an increasingly critical issue as more servers throughout the Internet are upgraded to Windows 2012 and IIS 8. ColdFusion is routinely used to make remote calls to other servers over SSL and it obviously needs to be updated to use this new standard. I suspect, given its convenience and implications, that many Windows server administrators will want to leverage this feature. As for our customer he is fine with not using SNI for the time being. His server is dedicated to this one task with only one domain hosted. Meanwhile I entered a bug in adobe's bug database at **this URL**. Please don't hesitate to vote it up.

Is there a "Fall Back" for SNI if a browser does not support it? Unfortunately not. But there is a work around for IIS 8 that requires you to setup a second site that does not have SNI enabled. In other words you have one site that handles SNI through the central repository and another site that works "the old way" and is bound to an individual IP address. Of course this appears to negate the whole point of SNI.

(The Muse writes)
Thanks to Wil for this excellent write-up and for his legwork tracking down the issue and resolution. I think we should invite him to write more in the near future. If you are a regular reader you probably know what I'm going to say next. This post references

Microsoft Windows. That's not an invitation for you to tell us why Apache is better or to castigate all the thousands of sys admins who use Windows and like it. On the other hand the Muse welcomes constructive comments on topic so feel free to chime in and add to our knowledge base.