ColdFusion and JVM Versions and SSLv3-TLS Security Magic

Posted At : December 8, 2014 2:52 PM | Posted By : Mark Kruger
Related Categories: ColdFusion, Coldfusion Security

This is the second entry by Wil Genovese (**Trunkful.com**) in our effort to provide a complete picture of how CF, Various versions of JVMs and various versions of SSL all work together. Wil's previous article on **Surviving Poodle** detailed a blow by blow description of how to troubleshoot a system broken due to the upgrading of SSL. This article includes some detailed technical information as well as the results of some painstaking tests. It is our hope that it will serve as a guide. It represents yet another reason to insure that you are upgrading to the latest JVM and CF version. Take it away Wil:

Wil writes...

We've all been taking steps lately to protect our computers and servers from the POODLE flaw in SSLv3. At CF Webtools we've been updating our servers in various hosting facilities to prevent the use of plain old SSLv3. As a reminder, there is base SSLv3, and SSLv3 plus TLS1.x. More about that later. Perhaps you never think about it, but as a ColdFusion developer you make frequent use SSL via various ColdFusion tags or cfscript. For example, CFHTTP lets you access a remote server (such as a web service) with a URL via ColdFusion server and it most often uses SSL in the process.

POODLE and ColdFusion

In case you missed why this is a trending topic (and why security folks like myself and the Muse are so riled up about it), here is a quick refresher as to what POODLE is according to **US-CERT**:

> "All systems and applications utilizing the Secure Socket Layer (SSL) 3.0 with cipher-block chaining (CBC) mode ciphers may be vulnerable. However, the POODLE (Padding Oracle On Downgraded Legacy Encryption) attack demonstrates this vulnerability using web browsers and web servers, which is one of the most likely exploitation scenarios. "
>
> "This affects most current browsers and websites, ***but also includes any software that either references a vulnerable SSL/TLS library*** (e.g. OpenSSL) or implements the SSL/TLS protocol suite itself."

In our case think of ColdFusion using CFHTTP to access a remote server over SSL. ColdFusion *becomes* the browser. Communications to the remote server (if vulnerable) could be compromised via **Man in the Middle attack**. This type of attack combined with the POODLE attack could lead to information exposer such as passwords or other authentication tokens that could then let an attacker access additional systems.

ColdFusion CFHTTP Under the Hood

How does ColdFusion create an HTTP request? CF uses an Apache Java class called *HttpClient* under the hood to do actual CFHTTP calls. The specific version was upgraded to Apache HttpClient 4.3.3 for ColdFusion 11. I found this information in the notes from a **bug report** I created for ColdFusion 10. ColdFusion 8, 9 and 10 all use an older version of this library so there is a notable difference in behaviors. The basic rule is that the older version of HttpClient does not appear to allow SSL fallback to older

SSL versions. Here's how I developed this data:

Testing Methodology

To test for each possible case scenarios I used this simple code:

```
<cfhttp url="https://accounts.google.com/ServiceLogin" method="GET" port="443">
<cfdump var="#cfhttp#">
```

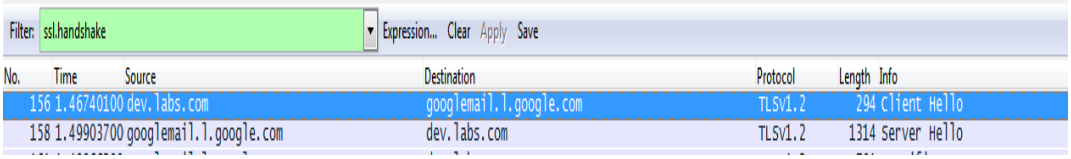
And tested it against the following ColdFusion versions (and yes, at CF Webtools we run and maintain all of these versions for various customers):

- ColdFusion 8.0.1 Fully Patched
- ColdFusion 9.0.2 - fresh unpachted install
- ColdFusion 10 Fully Patched
- ColdFusion 11 Update 3 (Prerelease)
- Java 1.6.0_04
- Java 1.6.0_45
- Java 1.7.0_71
- Java 1.8.0_25

From my research I am finding that Java 1.6 is only capable of SSLv3 and TLS1.0 this is based on this **article** at Oracle. The table below comes from that article.

	JDK 8 (March 2014 to present)	JDK 7 (July 2011 to present)	JDK 6 (2006 to end of public updates 2013)
TLS Protocols	TLSv1.2 (default) TLSv1.1 TLSv1 SSLv3	TLSv1.2 TLSv1.2 TLSv1 (default) SSLv3	TLSv1 (default) SSLv3
JSSE Ciphers:	Ciphers in JDK 8	Ciphers in JDK 7	Ciphers in JDK 6
Reference:	JDK 8 JSSE	JDK 7 JSSE	JDK 6 JSSE
Java Cryptography Extension, Unlimited Strength (explained later)	JCE for JDK 8	JCE for JDK 7	JCE for JDK 6

I setup each ColdFusion instance to use one of the specified Java versions for each round of testing. To verify the type of connection that was being made I used WireShark to intercept all network traffic and filtered for "ssl.handshake". This filter in Wireshark shows the exact handshake protocol that is used for the SSL connection. Here's a sample of wireshark output:



Based on the Oracle article noted above I used this Java arg in the jvm.config *-Dhttps.protocols*. This argument is supposed to specify the version of SSL/TLS allowed by Java. It can be used with any of the following values singly or in a comma separated list like so:

```
-Dhttps.protocols=TLSv1.1,TLSv1
```

The possible values are:

- SSLv3
- TLSv1
- TLSv1.1
- TLSv1.2

According the the chart above only certain SSL protocols are available for each different Java version. The table below shows my results of each test for each Java version vs ColdFusion Version vs the Java argument that I used for the test. The value in the chart indicates the results of the test based on the captured request data in WireShark.

	ColdFusion 8.0.1	ColdFusion 9.0.1	ColdFusion 10	ColdFusion 11
Java 1.6.0_04 SSLv3	TLSv1.0		TLSv1.0	
Java 1.6.0_45 SSLv3	TLSv1.0	TLSv1.0	TLSv1.0	
Java 1.6.0_45 TLSv1	TLSv1.0	TLSv1.0	TLSv1.0	
Java 1.6.0_45 TLSv1, SSLv3	TLSv1.0	TLSv1.0	TLSv1.0	
Java 1.7.0_71 SSLv3 Only		TLSv1.0	TLSv1.0	SSLv3
Java 1.7.0_71 TLSv1		TLSv1.0	TLSv1.0	TLSv1.0
Java 1.7.0_71 TLSv1, SSLv3		TLSv1.0	TLSv1.0	TLSv1.0
Java 1.7.0_71 TLSv1.2,TLSv1.1,TLSv1		TLSv1.0	TLSv1.0	TLSv1.2

Java 1.7.0_71 TLSv1.2,TLSv1.1		TLSv1.0	TLSv1.0	TLSv1.2
Java 1.8.0_25 SSLv3		TLSv1.2	TLSv1.2	SSLv3
Java 1.8.0_25 TLSv1		TLSv1.2	TLSv1.2	TLSv1.0
Java 1.8.0_25 TLSv1, SSLv3		TLSv1.2	TLSv1.2	TLSv1.0
Java 1.8.0_25 TLSv1.2,TLSv1.1,TLSv1		TLSv1.2	TLSv1.2	TLSv1.2

Findings

The most important finding is that no matter which value that I used for the *-Dhttps.protocols* argument with ColdFusion 8, 9 or 10 the resulting protocol that was used is the default protocol for that version of Java. In other words, for CF 8-10 this setting has no effect. Java 1.6 and 1.7 the default protocol is TLSv1.0 and I never could force CFHTTP to make a handshake with SSLv3. For Java 1.7 I never could get a higher protocol than TLSv1.0 to be used. For Java 1.8 the default protocol is TLSv1.2 and all attempts resulted in the SSL handshake using TLSv1.2 - again this is for ColdFusion 8 through 10.

ColdFusion 11 is where this story changes. CF 11 uses the newer Apache HttpClient. Consequently each result reflected the value of the *-Dhttps.protocols* argument. In cases where multiple protocols were given the highest possible protocol was used.

Conclusions

I think the following argument, *-Dhttps.protocols=TLSv1.2,TLSv1.1,TLSv1*, should be used to prevent SSL fallback to SSLv3 for ColdFusion 11. The older versions of ColdFusion do not need this argument because the default value for the given Java version is always used and that is never base SSLv3 (although interestingly you can *force* the use of base SSLv3 in CF 11 using the argument). The only caveat here is that Adobe may eventually update ColdFusion 10 with the newer Apache HttpClient. It is something we will certainly be pressing them to do. If they do then it will most likely be advisable to use the same argument settings for ColdFusion 10 in the future.

Interesting Side Note

Did you happen to notice the chart that I did get ColdFusion 9.0.2 to run on Java 1.8? That's interesting. I have not seen any notice from Adobe about there being an update for ColdFusion 9.0.2 to run on Java 1.8. I would not recommend trying this in production without extensive testing, but it's an interesting finding.

Muse Writes...

Thanks Wil! I know our readers will find this information incredibly valuable. If you are like me dear reader you tend to bookmark posts like this and return to them again and again as you stumble on to different environments. As always we welcome your comments and contributions to our compendium of knowledge.