According to wiki XML Signature,

When validating an XML Signature, a procedure called **Core Validation** is followed.

1. **Reference Validation:** Each Reference's digest is verified by retrieving the corresponding resource and applying any transforms and then the specified digest method to it. The result is compared to the recorded DigestValue; if they do not match, validation fails.
2. **Signature Validation:** The SignedInfo element is serialized using the canonicalization method specified in CanonicalizationMethod, the key data is retrieved using KeyInfo or by other means, and the signature is verified using the method specified in SignatureMethod.

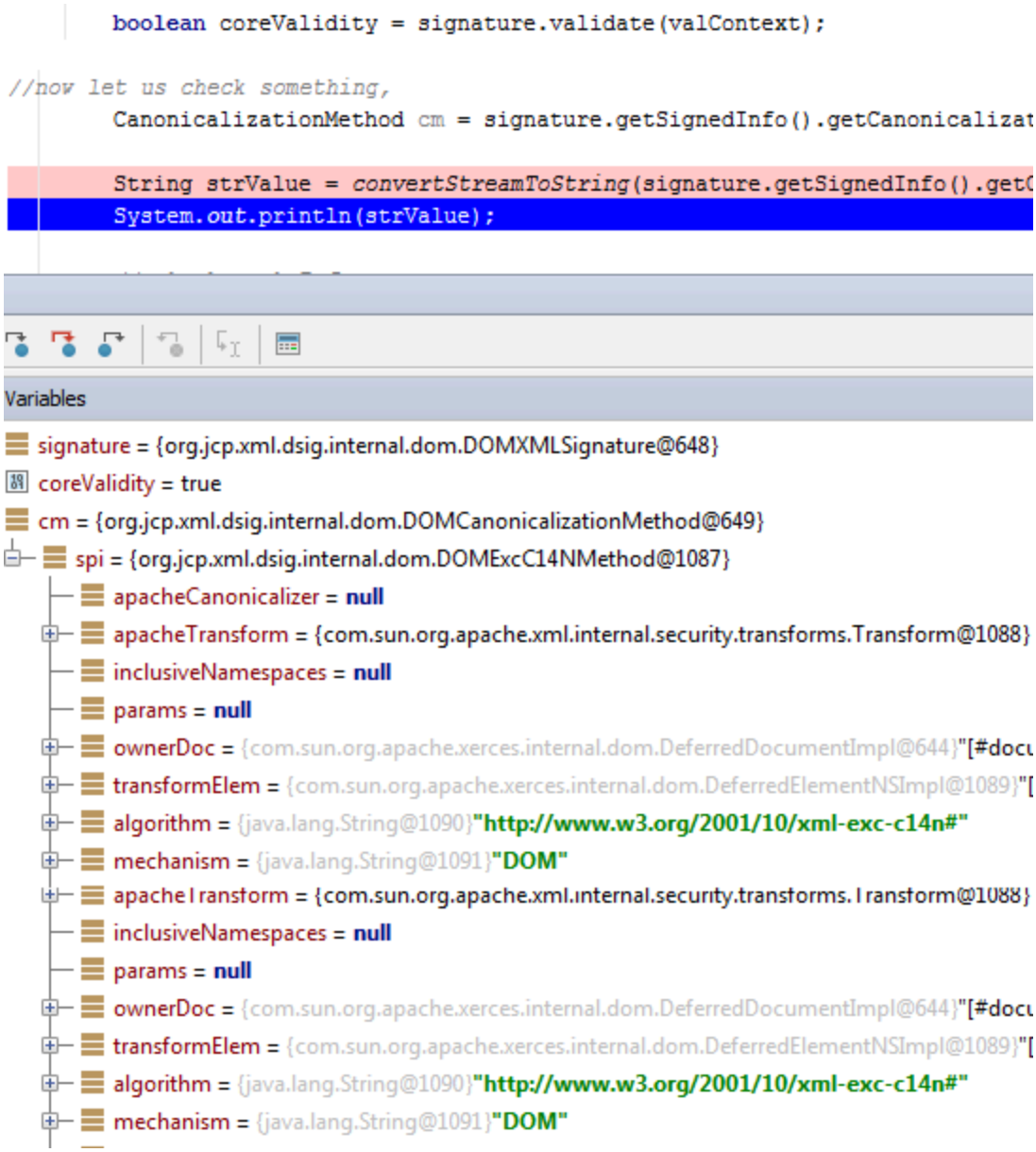
I am going to look at the procedure with pure Java and Apache Santuario utility *checksig*.

Note:
1, You can download the JAVA code and sample file.
2, Unlike 1.7.0, the checksig utility in 1.7.2 should take id parameter, for instance,

```
checksig  --id ID c:\dispute\saml_response.xml
```

First of all, let us try Java code,

As you can see, the core validation result *coreValidity* is **true**, so my sample passed the core validation.



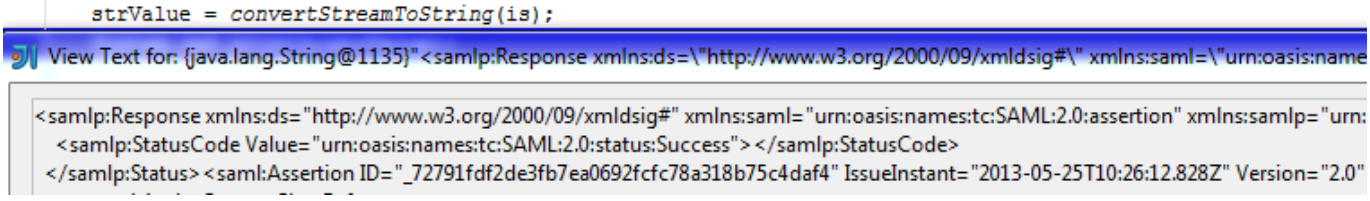
In addition, let us check the canonicalized bytes of the SignedInfo element, this is done by **xml-exc-c14n**.

You can compare with the original SignedInfo part, see the difference.

```
<ds:SignedInfo>
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<ds:Reference URI="#_9523e4bf9ad508763d904058d8eb52d45a57d3f8">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
</ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>OJsZTllFD8J5oFIsm/qJheEocUc=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
```

OK, let us check the Reference's pre-digested input stream, which is the main dispute between me and F5.

```
// check each Reference
Iterator it = signature.getSignedInfo().getReferences().iterator();
for (int k=0; it.hasNext(); k++) {
    Reference ref = (Reference) it.next();
    java.io.InputStream is = ref.getDigestInputStream();
    strValue = convertStreamToString(is);
```



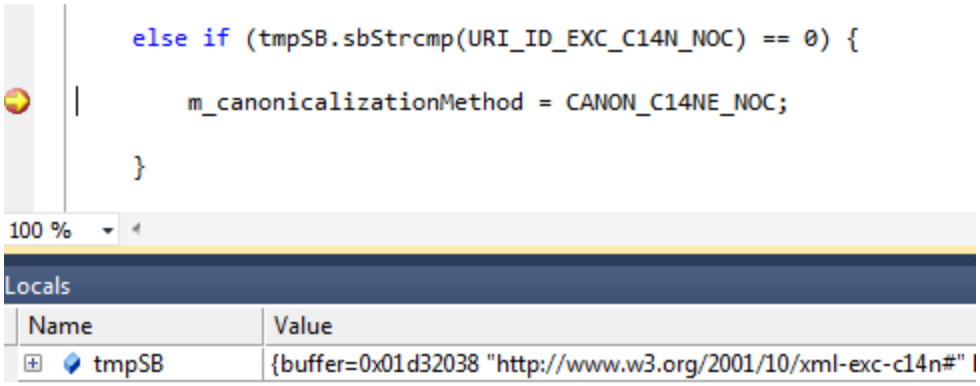
here we only have one reference, the screenshot only showed the beginning part.
This is the whole canonicalization result ,

```
<samlp:Response xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
Destination="https://bigip-sp.deepnetsecurity.local/saml/sp/profile/post/acs"
ID="_9523e4bf9ad508763d904058d8eb52d45a57d3f8" IssueInstant="2013-05-25T10:26:12.828Z"
Version="2.0"><saml2:Issuer
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">urn:deepnet:dual:idp:appsso</saml2:Issuer>
<samlp:Status>
    <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"></samlp:StatusCode>
  </samlp:Status><saml:Assertion ID="_72791fdf2de3fb7ea0692fcfc78a318b75c4daf4"
IssueInstant="2013-05-25T10:26:12.828Z" Version="2.0"><saml2:Issuer
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">urn:deepnet:dual:idp:appsso</saml2:Issuer>
<saml:Subject><saml2:NameID xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">dlucas</saml2:NameID><saml:Sub
jectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer"><saml:SubjectConfirmationData
InResponseTo="_d5ab6cac57ca2b7cc70845457f6ed000d13296" NotOnOrAfter="2023-05-25T10:26:13.156Z"
Recipient="https://bigip-sp.deepnetsecurity.local/saml/sp/profile/post/acs"></saml:SubjectConf
irmationData></saml:SubjectConfirmation></saml:Subject><saml:Conditions
NotBefore="2013-05-25T10:26:12.828Z"
NotOnOrAfter="2023-05-25T10:26:12.828Z"><saml:AudienceRestriction><saml:Audience>https://bigip
-sp.deepnetsecurity.local</saml:Audience></saml:AudienceRestriction></saml:Conditions><saml:Au
thnStatement AuthnInstant="2013-05-25T10:26:12.828Z"
SessionNotOnOrAfter="2023-05-25T10:26:12.828Z"><saml:AuthnContext>
           <saml:AuthnContextClassRef>
             urn:oasis:names:tc:SAML:2.0:ac:classes:Password
           </saml:AuthnContextClassRef>
      </saml:AuthnContext></saml:AuthnStatement><saml:AttributeStatement><saml2:Attribute
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" Name="dasDomainName"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml2:AttributeValue>tivoli</sam
l2:AttributeValue></saml2:Attribute><saml2:Attribute
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" Name="lastLogin"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml2:AttributeValue>2013-05-25T
10:26:12Z</saml2:AttributeValue></saml2:Attribute><saml2:Attribute
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" Name="canonicalName"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml2:AttributeValue>dlucas</sam
l2:AttributeValue></saml2:Attribute><saml2:Attribute
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" Name="profileId"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml2:AttributeValue>13694775544
37182</saml2:AttributeValue></saml2:Attribute><saml2:Attribute
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" Name="userId"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml2:AttributeValue>#eu#_User_2
9_Diana Lucas_null</saml2:AttributeValue></saml2:Attribute><saml2:Attribute
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" Name="fullName"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml2:AttributeValue>Lucas,null<
/saml2:AttributeValue></saml2:Attribute><saml2:Attribute
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" Name="loginName"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml2:AttributeValue>dlucas</sam
l2:AttributeValue></saml2:Attribute></saml:AttributeStatement></saml:Assertion></samlp:Respons
e>
```

As you can see, it is identical to Apache's result, but it is different from F5's result, you can go back Part 3 to compare these results.

Now, let us look at how apache utility *checksig* does the job on the same sample

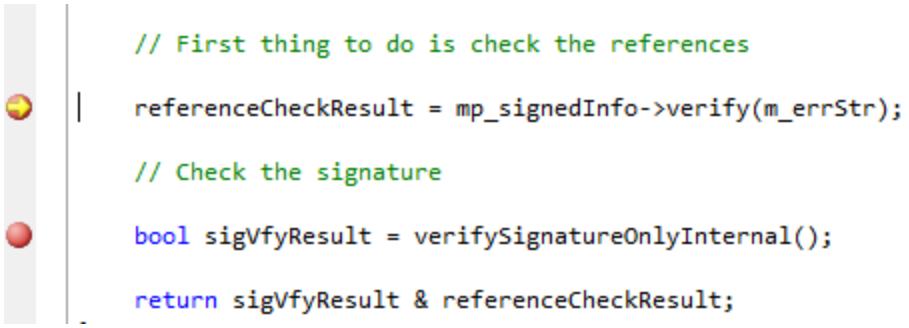
file.



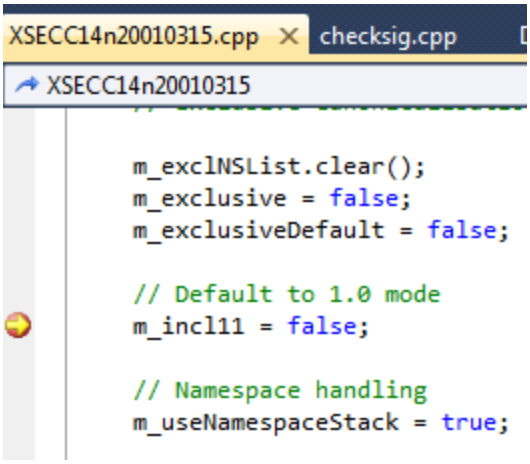

```
#define URI_ID_C14N11_NOC       "http://www.w3.org/2006/12/xml-c14n11"
#define URI_ID_EXC_C14N_NOC     "http://www.w3.org/2001/10/xml-exc-c14n#"
```

Look it carefully, F5 doubted I was using `URI_ID_C14N11_NOC`, actually I wasn't.

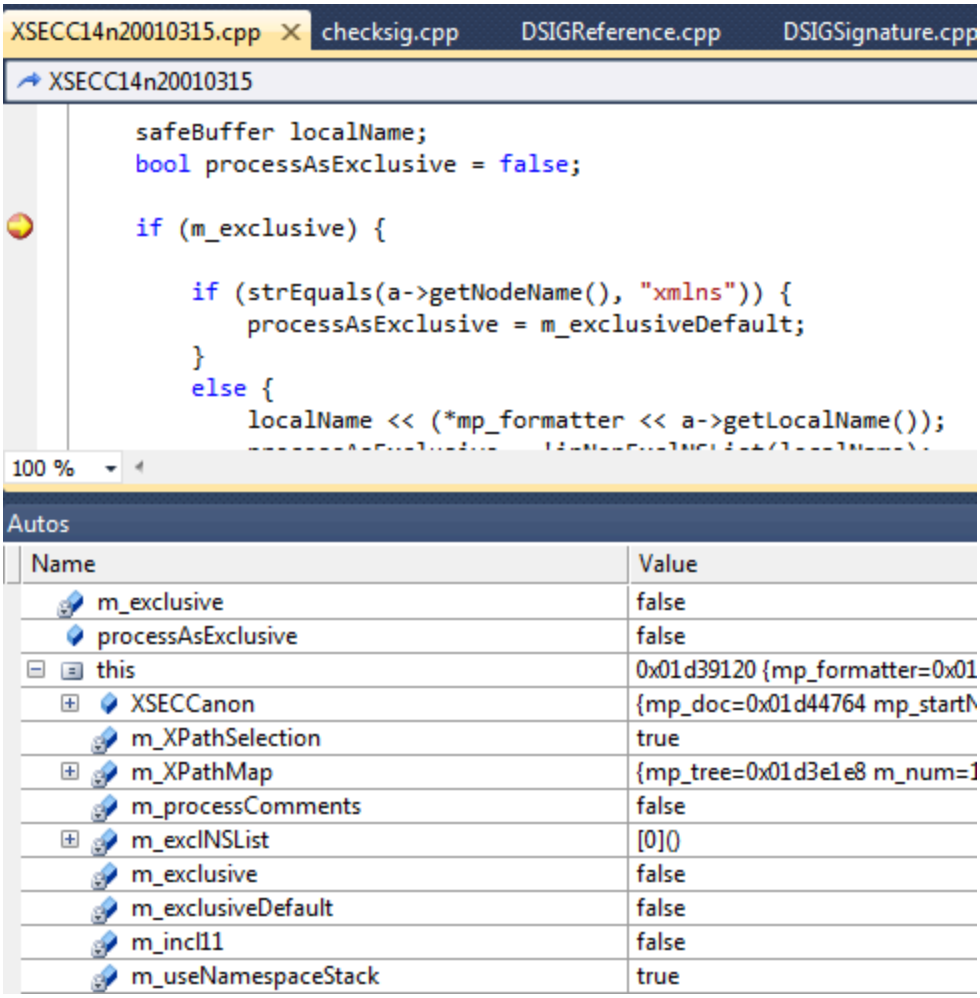
Let us move on, we see the whole validation is composed of two steps.



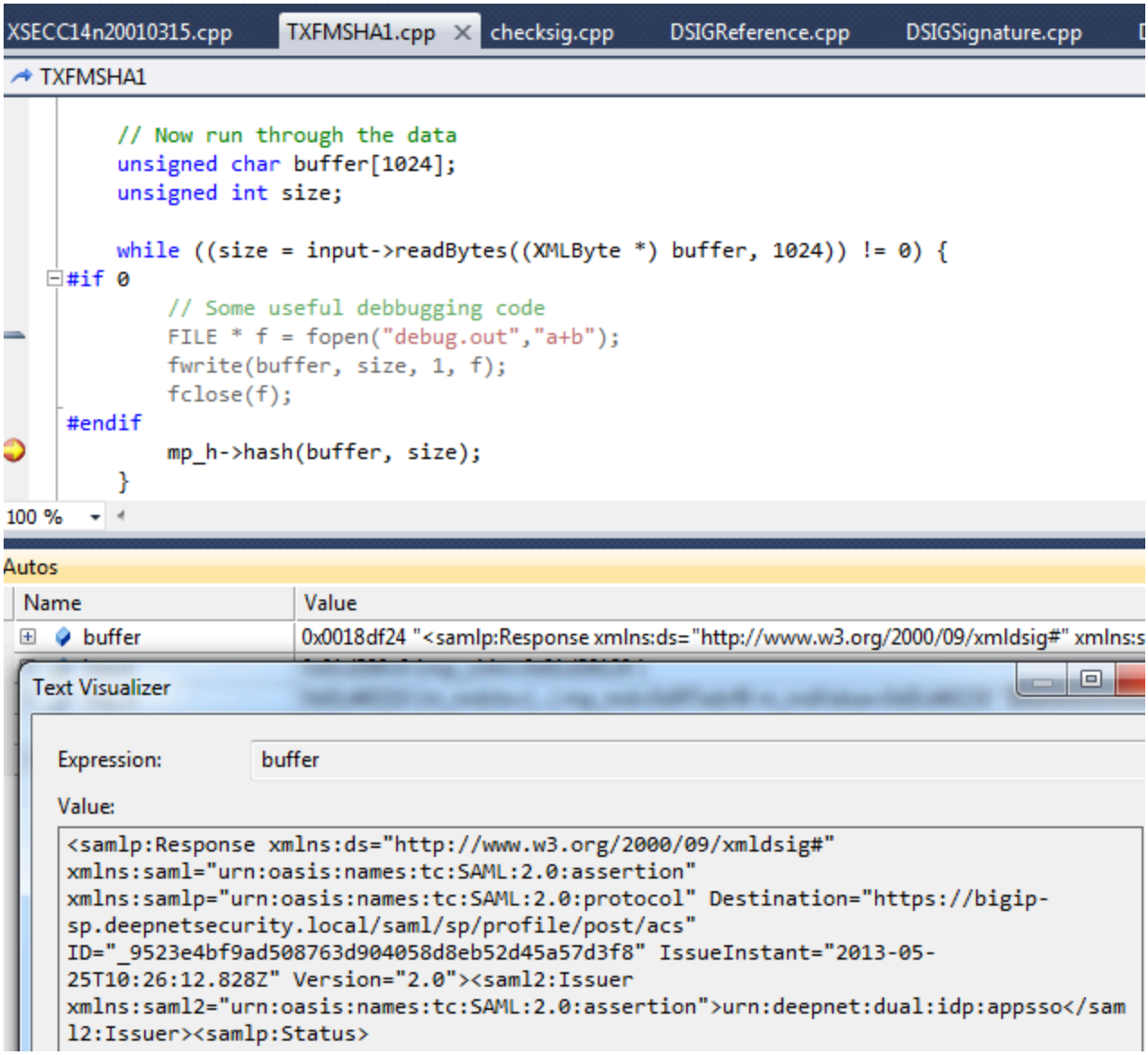
First, it tries to verify reference(s). We set up a breakpoint in XSECC14n20010315.cpp, you can see, exclusive = **false** by the default.



It uses the default XML canonicalization C14N which is **NOT** exclusive on reference handling.

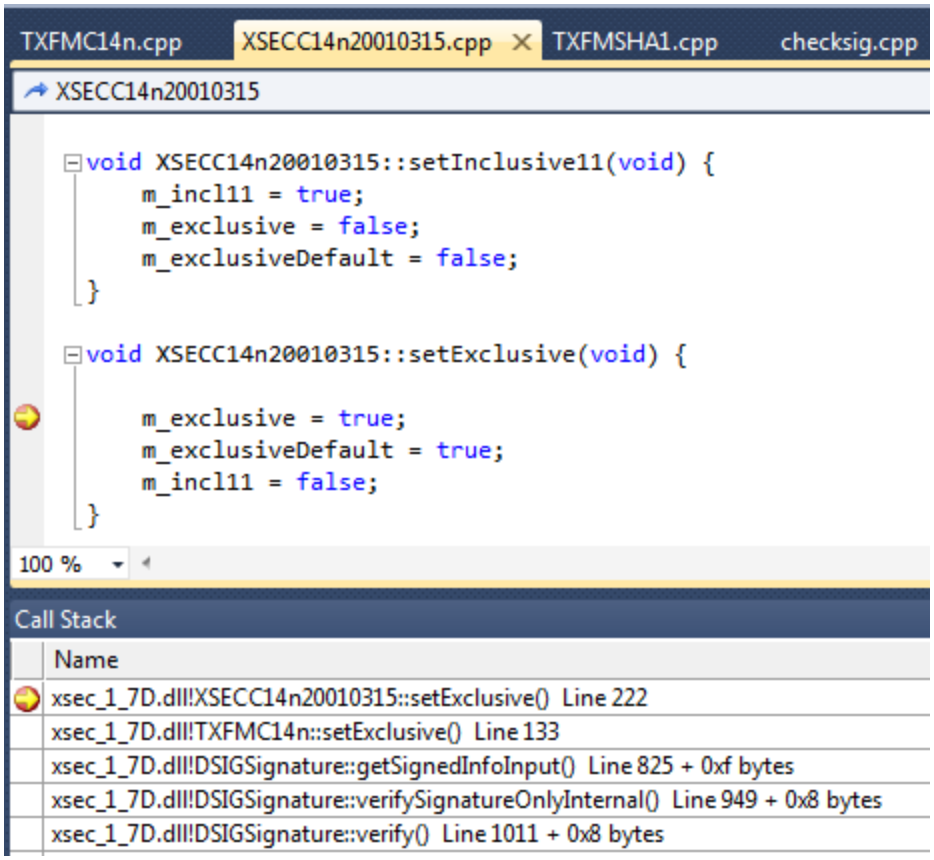


Furthermore, check the canonical result, the header part is as same as the Java result.

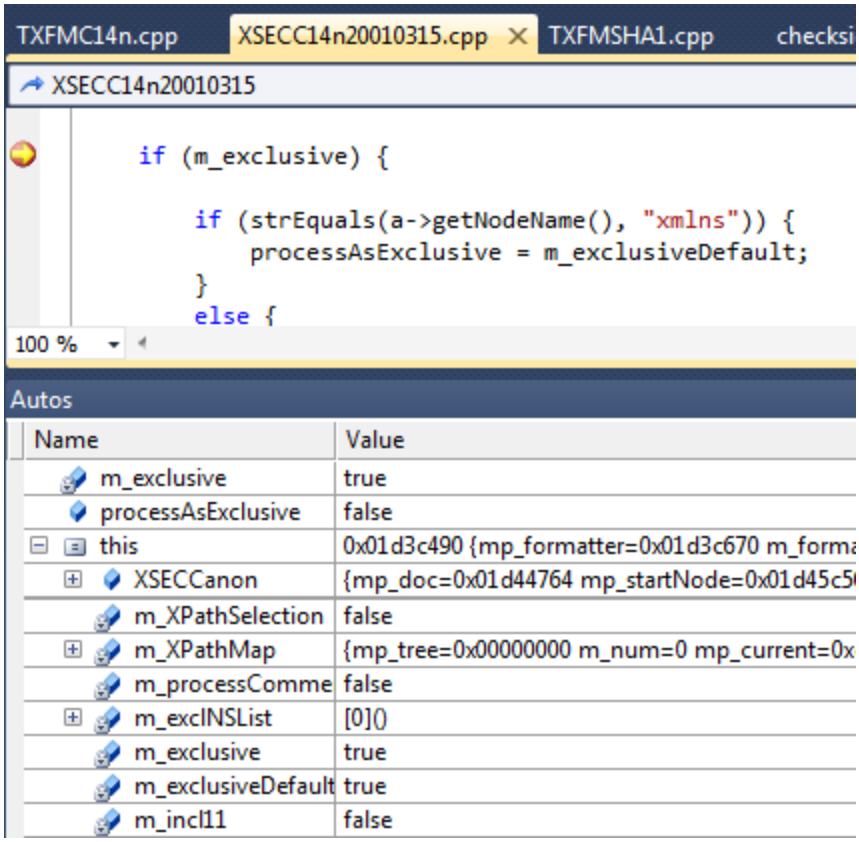


Let us continue to process the signature validation (second part). In call stack, you can see when in **verifySignatureOnlyInternal,** the exclusive settings turned ON. Why? because we specify the canonicalization method in SignedInfo.

```
<ds:SignedInfo>
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
```



Again, check the exclusive value, who now is **true**.



Conclusion:

The CanonicalizationMethod (in my case, "http://www.w3.org/2001/10/xml-exc-c14n#") defined in SignedInfo should be **ONLY** used on the second step of core validation - *Signature Validation*.
However, F5 also employed the same method (the exclusive one) onto the first step (*Reference Validation*), which I think is wrong.