

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.

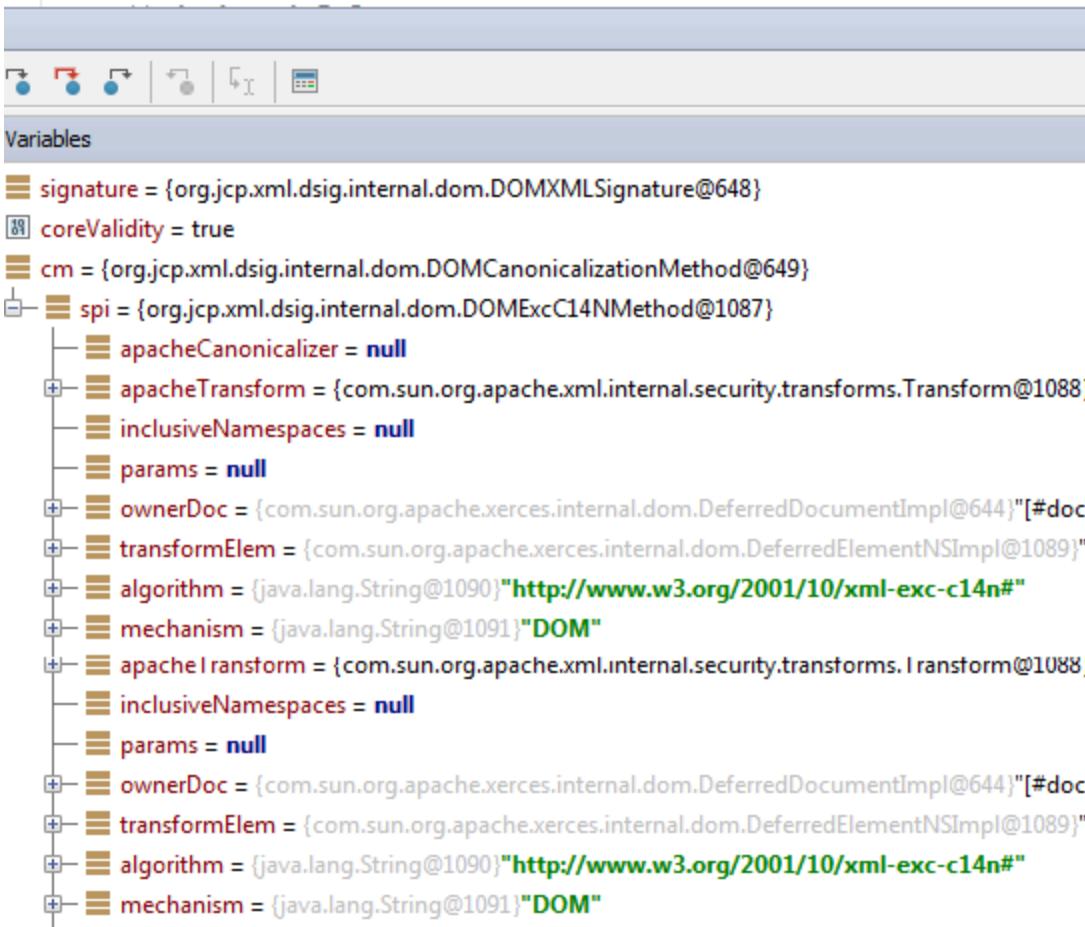
```

        boolean coreValidity = signature.validate(valContext);

//now let us check something,
    CanonicalizationMethod cm = signature.getSignedInfo().getCanonicalizat

    String strValue = convertStreamToString(signature.getSignedInfo().getO
    System.out.println(strValue);

```



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

View Text for: (java.lang.String@1083)"<ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">\n<d...

```

<ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<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>OJsZTl1FD8J5oFIsm/qJheEocUc=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>

```

Close

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>OJsZTl1FD8J5oFIsm/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);

```

View Text for: (java.lang.String@1135)"<samlp:Response xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" xmlns:samlp="urn:...

```

<samlp:Response xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" xmlns:samlp="urn:...
<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"

```

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

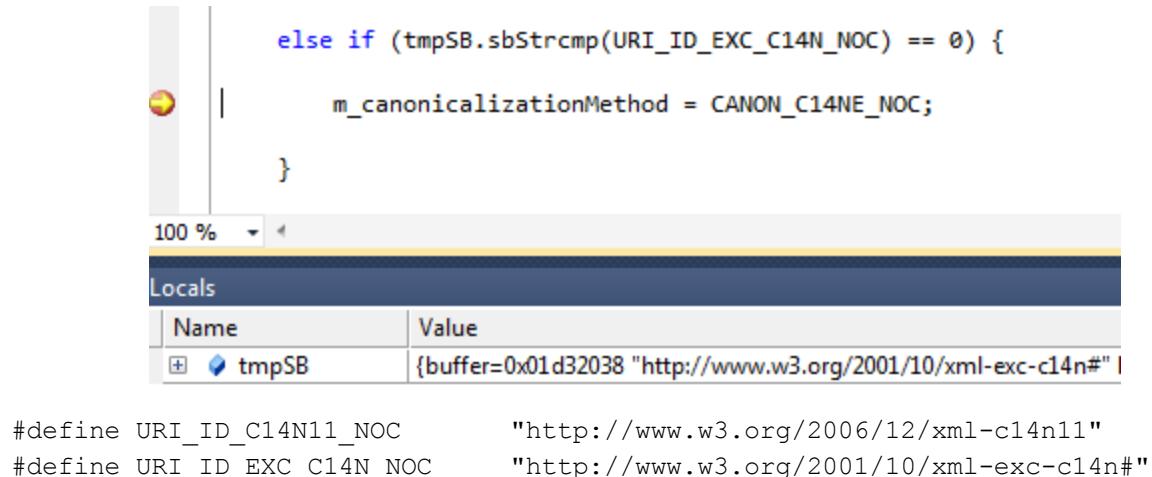
```

<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:appssso</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:appssso</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">ducas</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
        irminationData></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>ducas</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>ducas</sam
        l2:AttributeValue></saml2:Attribute></saml:AttributeStatement></saml:Assertion></samlp:Respon
        se>
```

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.



The screenshot shows a debugger interface with a code editor and a 'Locals' window. The code editor displays:

```
else if (tmpSB.sbStrcmp(URI_ID_EXC_C14N_NOC) == 0) {  
    m_canonicalizationMethod = CANON_C14NE_NOC;  
}  
  
100 %
```

The 'Locals' window shows a single entry:

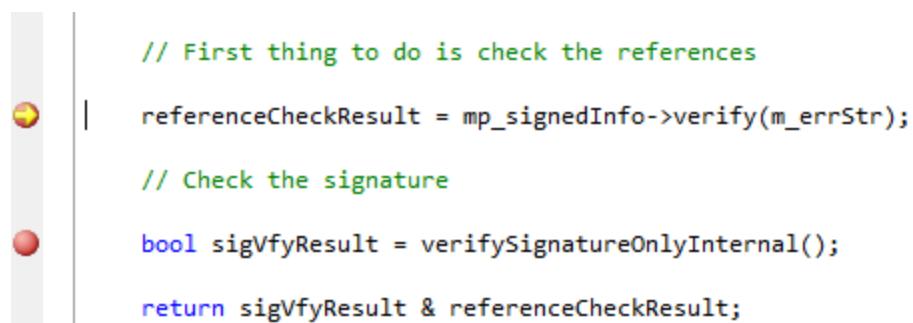
Name	Value
tmpSB	{buffer=0x01d32038 "http://www.w3.org/2001/10/xml-exc-c14n#"}

Below the code editor, there are two preprocessor definitions:

```
#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.

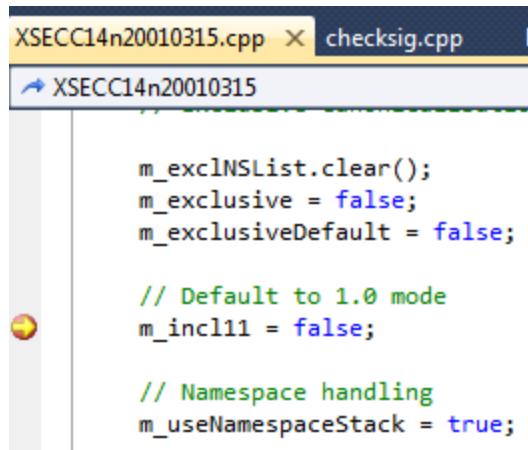
Let us move on, we see the whole validation is composed of two steps.



The screenshot shows a debugger interface with a code editor and a 'Locals' window. The code editor displays:

```
// First thing to do is check the references  
referenceCheckResult = mp_signedInfo->verify(m_errStr);  
  
// Check the signature  
bool sigVfyResult = verifySignatureOnlyInternal();  
  
return sigVfyResult & referenceCheckResult;
```

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



The screenshot shows a debugger interface with a code editor and a 'Locals' window. The code editor displays:

```
XSECC14n20010315.cpp X checksig.cpp  
XSECC14n20010315  
  
m_exclNSList.clear();  
m_exclusive = false;  
m_exclusiveDefault = false;  
  
// Default to 1.0 mode  
m_incl11 = false;  
  
// Namespace handling  
m_useNamespaceStack = true;
```

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

The screenshot shows a debugger interface with several tabs at the top: XSECC14n20010315.cpp (selected), checksig.cpp, DSIGReference.cpp, and DSIGSignature.cpp. Below the tabs, there is a code editor window displaying C++ code related to XML processing. In the bottom right corner of the code editor, there is a small yellow circular icon with a red exclamation mark, indicating a warning or error. Below the code editor is an 'Autos' table showing variable values:

Name	Value
m_exclusive	false
processAsExclusive	false
this	0x01d39120 {mp_formatter=0x01}
XSECCanon	{mp_doc=0x01d44764 mp_startNode=0x01d39120}
m_XPathSelection	true
m_XPathMap	{mp_tree=0x01d3e1e8 m_num=1}
m_processComments	false
m_exclNSList	[0]()
m_exclusive	false
m_exclusiveDefault	false
m_incl11	false
m_useNamespaceStack	true

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

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar has tabs for XSECC14n20010315.cpp, TXFMSHA1.cpp (which is the active tab), checksig.cpp, DSIGReference.cpp, and DSIGSignature.cpp. Below the tabs, there is a toolbar with icons for back, forward, and search.

The main code editor window displays the TXFMSHA1.cpp file. The code is as follows:

```

    // Now run through the data
    unsigned char buffer[1024];
    unsigned int size;

    while ((size = input->readBytes((XMLByte *) buffer, 1024)) != 0) {
        #if 0
            // Some useful debugging code
            FILE * f = fopen("debug.out", "a+b");
            fwrite(buffer, size, 1, f);
            fclose(f);
        #endif
        mp_h->hash(buffer, size);
    }

```

The 'Autos' window below the code editor shows a table with one row:

Name	Value
buffer	0x0018df24 "<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:appssso</saml2:Issuer><samlp:Status>"

The 'Text Visualizer' window shows the XML content of the buffer variable:

```

Expression: buffer
Value:
<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:appssso</saml2:Issuer><samlp:Status>
```

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#" />
```

The screenshot shows a debugger interface with several tabs at the top: TXFMC14n.cpp, XSECC14n20010315.cpp (highlighted in yellow), TXFMSHA1.cpp, and checksig.cpp. Below the tabs, the assembly code for XSECC14n20010315 is displayed. Two functions are shown:

```
void XSECC14n20010315::setInclusive11(void) {
    m_incl11 = true;
    m_exclusive = false;
    m_exclusiveDefault = false;
}

void XSECC14n20010315::setExclusive(void) {
    m_exclusive = true;
    m_exclusiveDefault = true;
    m_incl11 = false;
}
```

Below the code, a call stack window titled "Call Stack" is open, showing the following stack trace:

Name
xsec_1_7D.dll!XSECC14n20010315::setExclusive() Line 222
xsec_1_7D.dll!TXFMC14n::setExclusive() Line 133
xsec_1_7D.dll!DSIGSignature::getSignedInfoInput() Line 825 + 0xf bytes
xsec_1_7D.dll!DSIGSignature::verifySignatureOnlyInternal() Line 949 + 0x8 bytes
xsec_1_7D.dll!DSIGSignature::verify() Line 1011 + 0x8 bytes

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

The screenshot shows a debugger interface with several tabs at the top: TXFMC14n.cpp, XSECC14n20010315.cpp (highlighted in yellow), TXFMSHA1.cpp, and checksi. Below the tabs is a code editor window titled 'XSECC14n20010315' containing C++ code. The code snippet is as follows:

```

if (m_exclusive) {

    if (strEquals(a->getNodeName(), "xmlns")) {
        processAsExclusive = m_exclusiveDefault;
    }
    else {

```

Below the code editor is an 'Autos' window displaying variable values:

Name	Value
m_exclusive	true
processAsExclusive	false
this	0x01d3c490 {mp_formatter=0x01d3c670 m_formatter=0x01d3c670}
XSECCanon	{mp_doc=0x01d44764 mp_startNode=0x01d45c51}
m_XPathSelection	false
m_XPathMap	{mp_tree=0x00000000 m_num=0 mp_current=0x00000000}
m_processComme	false
m_exclNSList	[0]()
m_exclusive	true
m_exclusiveDefault	true
m_incl11	false

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.