Duo Finds SAML Vulnerabilities Affecting Multiple Implementations

Duo Finds SAML Vulnerabilities Affecting Multiple Implementations - Kelby Ludwig, Duo Security.

- Published: date not stated
- Original: <https://duo.com/blog/duo-finds-saml-vulnerabilities-affecting-multiple-implementations>
- Preserved from: https://duo.com/blog/duo-finds-saml-vulnerabilities-affecting-multiple-implementations (stored) on 2026-08-14
- Capture timestamp: 20181116121859
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Duo Labs

February 27th, 2018 Kelby Ludwig

Duo Finds SAML Vulnerabilities Affecting Multiple Implementations

This blog post describes a new vulnerability class that affects SAML-based single sign-on (SSO) systems. This vulnerability can allow an attacker with authenticated access to trick SAML systems into authenticating as a different user without knowledge of the victim user's password.

Duo Labs, the advanced research team of Duo Security, has identified multiple vendors that were affected by this flaw:

- OneLogin - python-saml - CVE-2017-11427
- OneLogin - ruby-saml - CVE-2017-11428
- Clever - saml2-js - CVE-2017-11429
- OmniAuth-SAML - CVE-2017-11430
- Shibboleth - CVE-2018-0489
- Duo Network Gateway - CVE-2018-7340

We recommend that individuals that rely on SAML-based SSO to update any affected software to patch this vulnerability. If you are a Duo Security customer running Duo Network Gateway (DNG),

please see our Product Security Advisory here.

SAML Responses, Briefly

The Security Assertion Markup Language, SAML, is a popular standard used in single sign-on systems. Greg Seador has written a great pedagogical guide on SAML that I highly recommend if you aren't familiar with it.

For the purpose of introducing this vulnerability, the most important concept to grasp is what a SAML `Response` means to a Service Provider (SP), and how it is processed. `Response` processing has a lot of subtleties, but a simplified version often looks like:

-

The user authenticates to an Identity Provider (IdP) such as Duo or GSuite which generates a signed SAML `Response`. The user's browser then forwards this response along to an SP such as Slack or Github.

-

The SP validates the SAML `Response`s signature.

-

If the signature is valid, a string identifier within the SAML Response (e.g. the `NameID`) will identify which user to authenticate.

A really simplified SAML `Response` could look something like:

```
<SAMLResponse>
  <Issuer>https://idp.com/</Issuer>
  <Assertion ID="_id1234">
    <Subject>
      <NameID>user@user.com</NameID>
    </Subject>
  </Assertion>
  <Signature>
    <SignedInfo>
      <CanonicalizationMethod Algorithm="xml-c14n11"/>
      <Reference URI="#_id1234"/>
    </SignedInfo>
    <SignatureValue>
      some base64 data that represents the signature of the assertion
    </SignatureValue>
  </Signature>
</SAMLResponse>
```

This example omits a lot of information, but that omitted information is not too important for this vulnerability. The two essential elements from the above XML blob are the `Assertion` and the

`Signature` element. The `Assertion` element is ultimately saying "Hey, I, the Identity Provider, authenticated the user `user@user.com`." A signature is generated for that `Assertion` element and stored as part of the `Signature` element.

The `Signature` element, if done correctly, should prevent modification of the `NameID`. Since the SP likely uses the `NameID` to determine what user should be authenticated, the signature prevents an attacker from changing their own assertion with the `NameID` "attacker@user.com" to "user@user.com." If an attacker can modify the `NameID` without invalidating the signature, that would be bad (hint, hint)!

XML Canononononicalizizization: Easier Spelt Than Done

The next relevant aspect of XML signatures is XML canonicalization. XML canonicalization allows two logically equivalent XML documents to have the same byte representation. For example:

```
<A X="1" Y="2">some text<!-- and a comment --></A>
```

and

```
< A Y="2" X="1" >some text</ A >
```

These two documents have different byte representations, but convey the same information (i.e. they are logically equivalent).

Canonicalization is applied to XML elements prior to signing. This prevents practically meaningless differences in the XML document from leading to different digital signatures. This is an important point so I'll emphasize it here: multiple different-but-similar XML documents can have the same exact signature. This is fine, for the most part, as what differences matter are specified by the canonicalization algorithm.

As you might have noticed in the toy SAML Response above, the `CanonicalizationMethod` specifies which canonicalization method to apply prior to signing the document. There are a couple of algorithms outlined in the XML Signature specification, but the most common algorithm in practice seems to be `http://www.w3.org/2001/10/xml-exc-c14n#` (which I'll just shorten to `exc-c14n`).

There is a variant of `exc-c14n` that has the identifier `http://www.w3.org/2001/10/xml-exc-c14n#WithComments`. This variation of `exc-c14n` does not omit comments, so the two XML documents above would *not* have the same canonical representation. This distinction between the two algorithms will be important later.

XML APIs: One Tree; Many Ways

One of the causes of this vulnerability is a subtle and arguably unexpected behavior of XML libraries like Python's `lxml` or Ruby's `REXML`. Consider the following XML element, `NameID`:

```
<NameID>kludwig</NameID>
```

And if you wanted to extract the user identifier from that element, in Python, you may do the following:

```
from defusedxml.lxml import fromstring
payload = "<NameID>kludwig</NameID>"
data = fromstring(payload)
return data.text # should return 'kludwig'
```

Makes sense, right? The `.text` method extracts the text of the `NameID` element.

Now, what happens if I switch things up a bit, and add a comment to this element:

```
from defusedxml.lxml import fromstring
doc = "<NameID>klud<!-- a comment? -->wig</NameID>"
data = fromstring(payload)
return data.text # should return ‘kludwig’?
```

If you would expect the exact same result regardless of the comment addition, I think you are in the same boat as me and many others. However, the `.text` API in `lxml` returns `klud`! Why is that?

Well, I think what `lxml` is doing here is *technically* correct, albeit a bit unintuitive. If you think of the XML document as a tree, the XML document looks like:

```
element: NameID
|_ text: klud
|_ comment: a comment?
|_ text: wig
```

and `lxml` is just not reading text after the first text node ends. Compare that with the uncommented node which would be represented by:

```
element: NameID
|_ text: kludwig
```

Stopping at the first text node in this case makes perfect sense!

Another XML parsing library that exhibits similar behavior is Ruby's `REXML`. The documentation for their `get_text` method hints at why these XML APIs exhibit this behavior:

```
[get_text] returns the first child Text node, if any, or nil otherwise. This method returns the actual Text node, rather th
```

Stopping text extraction after the first child, while unintuitive, might be fine if all XML APIs behaved this way. Unfortunately, this is not the case, and some XML libraries have nearly identical APIs but handle text extraction differently:

```
import xml.etree.ElementTree as et
doc = "<NameID>klud<!-- a comment? -->wig</NameID>"
data = et.fromstring(payload)
return data.text # returns 'kludwig'
```

I have also seen a few implementations that don't leverage an XML API, but do text extraction manually by just extracting the inner text of a node's first child. This is just another path to the same exact substring text extraction behavior.

The Vulnerability

So now we have the three ingredients that enable this vulnerability:

-

SAML Responses contain strings that identify the authenticating user.

-

XML canonicalization (in most cases) will remove comments as part of signature validation, so adding comments to a SAML Response will not invalidate the signature.

-

XML text extraction may only return a substring of the text within an XML element when comments are present.

So, as an attacker with access to the account `user@user.com.evil.com`, I can modify *my own* SAML assertions to change the NameID to `user@user.com` when processed by the SP. Now with a simple seven-character addition to the previous toy SAML Response, we have our payload:

```
<SAMLResponse>
  <Issuer>https://idp.com/</Issuer>
  <Assertion ID="_id1234">
    <Subject>
      <NameID>user@user.com<!---->.evil.com</NameID>
    </Subject>
  </Assertion>
  <Signature>
    <SignedInfo>
      <CanonicalizationMethod Algorithm="xml-c14n11"/>
      <Reference URI="#_id1234"/>
    </SignedInfo>
    <SignatureValue>
      some base64 data that represents the signature of the assertion
    </SignatureValue>
  </Signature>
</SAMLResponse>
```

How Does This Affect Services That Rely on SAML?

Now for the fun part: it varies greatly!

The presence of this behavior is not great, but not always exploitable. SAML IdPs and SPs are generally very configurable, so there is lots of room for increasing or decreasing impact.

For example, SAML SPs that use email addresses and validate their domain against a whitelist are much less likely to be exploitable than SPs that allow arbitrary strings as user identifiers.

On the IdP side, openly allowing users to register accounts is one way to increase the impact of this issue. A manual user provisioning process may add a barrier to entry that makes exploitation a bit more infeasible.

Remediation

Remediation of this issue somewhat depends on what relationship you have with SAML.

For Users of Duo's Software

Duo has released updates for the Duo Network Gateway in **version 1.2.10**. If you use the DNG as a SAML Service Provider and are not at version 1.2.10 or higher (at the time of writing this, 1.2.10 is the latest version), we recommend upgrading.

Learn more in Duo's Product Security Advisory (PSA) for this vulnerability.

If You Run or Maintain an Identity Provider or Service Provider

The best remediation is to ensure your SAML processing libraries are not affected by this issue. We identified several SAML libraries that either leveraged these unintuitive XML APIs or did faulty manual text extraction, but I'm sure there are more libraries out there that don't handle comments in XML nodes well.

Another possible remediation could be defaulting to a canonicalization algorithm such as `http://www.w3.org/2001/10/xml-exc-c14n#WithComments` which does not omit comments during canonicalization. This canonicalization algorithm would cause comments added by an attacker to invalidate the signature, but the canonicalization algorithm identifier itself must not be subject to tampering. This modification, however, would require IdP and SP support, which may not be universal.

Additionally, if your SAML Service Provider enforces two-factor authentication, that helps a lot because this vulnerability would only allow a bypass of a user's first factor of authentication. Note that if your IdP is responsible for both first factor and second factor authentication, it's likely that this vulnerability bypasses both!

If You Maintain a SAML Processing Library

The most obvious remediation here is ensuring your SAML library is extracting the full text of a given XML element when comments are present. Most SAML libraries I found had some form of unit tests, and it was fairly easy to update the tests which extracted properties like `NameID`s and just add comments to pre-signed documents. If the tests continue to pass, great! Otherwise, you may be vulnerable.

Another possible remediation is updating libraries to use the canonicalized XML document after signature validation for any processing such as text extraction. This could prevent this vulnerability as well as other vulnerabilities that could be introduced by XML canonicalization issues.

If You Maintain an XML Parsing Library

Personally, I think the number of libraries affected by this vulnerability suggest that many users also seem to assume XML inner text APIs are not affected by comments, so that could be a motivating factor to change an API's behavior. However, I don't think there is a clear right answer for XML library authors, and a very reasonable action may be keeping the APIs as they are and improving documentation surrounding this behavior.

Another possible remediation path is improving the XML standards. Through my research, I did not identify any standards that specified the correct behavior, and it may be worth specifying how these related standards should interoperate.

Disclosure Timeline

Our disclosure policy can be found here https://www.duo.com/labs/disclosure. In this case, due to the vulnerability impacting multiple vendors, we decided to work with CERT/CC to coordinate disclosure. The following is a high-level disclosure timeline:

| Date | Activity | | | 2017-12-18 | CERT/CC contacted and provided vulnerability information. | | | 2017-12-20 | CERT/CC follows up with questions about the issue. | | | 2017-12-22 | Response to

questions from CERT/CC. | | | 2018-01-02 to 2018-01-09 | Additional email discussion with CERT/CC about the issue. | | | 2018-01-24 | CERT/CC completes internal analysis and contacts impacted vendors. | | | 2018-01-25 | Vendors acknowledge CERT/CC report. Additional communication with CERT/CC and vendors to further explain the issue and other attack vectors. | | | 2018-01-29 | Additional vendor that is potentially impacted identified and contacted by CERT/CC. | | | 2018-02-01 | Duo Labs reserves CVE #s for each impacted vendor. | | | 2018-02-06 | Draft of CERT/CC vulnerability technical note reviewed and approved by Duo. | | | 2018-02-20 | Final confirmation that all impacted vendors are ready for disclosure date. | | | 2018-02-27 | Disclosure. | |

We would like to thank CERT/CC for helping us disclose this vulnerability and we appreciate all of the efforts put forth by each and everyone who was contacted by CERT/CC to quickly respond to this issue.
