Source Code Analysis and Penetration Test Report:

openconext-saml-java

Version: 1.0.1
15.01.2024

Juraj Somorovsky
Phone: +49(0)234 / 54459996 | E-Mail: Juraj.Somorovsky@hackmanit.de

Project Information

Customer:	SURF BV Postbus 19035 3501 Utrecht, Netherlands
Contact:	Peter Havekes
Commissioned to:	Hackmanit GmbH Universitätsstraße 60 (Exzenterhaus) 44789 Bochum, Germany
Project executive:	Juraj Somorovsky Phone: +49(0)234 / 54459996 Fax: +49(0)234 / 54427593 E-Mail: Juraj.Somorovsky@hackmanit.de
Project members:	Karsten Meyer zu Selhausen (Hackmanit GmbH)
Project period:	11.12.2023 – 19.12.2023
Version of the report:	1.0.1

This report was technically verified by Juraj Somorovsky.
This report was linguistically verified by Karsten Meyer zu Selhausen.

Hackmanit GmbH
Represented by: Prof. Dr. Jörg Schwenk, Prof. Dr. Juraj Somorovsky,
Dr. Christian Mainka, Prof. Dr. Marcus Niemietz
Register court: Amtsgericht Bochum, HRB 14896, Germany

Contents

1 Summary

Hackmanit GmbH was commissioned by SURF to perform a source code analysis and penetration test of their open-source library “openconext-saml-java” and an identity provider (IdP) test implementation based on the library. The source code analysis and penetration test were performed remotely with a total expense of four man-days – including documentation and writing of this report.

Weaknesses. During our penetration test, one weakness was identified and classified as *Medium*. The weakness M01 allows an attacker to execute an XML signature wrapping (XSW) attack on SAML authentication requests, which are processed by the IdP. With this attack, it is possible to arbitrarily modify the authentication request’s contents even if they are protected by XML Signatures.

The impact of this attack on SAML authentication requests is rather small; SAML authentication requests typically only contain data, which are defined in the IdP metadata. Nevertheless, relying on a vulnerable XML Signature validation could introduce unrestricted trust in the message content and result in weaknesses in future deployments.

Recommended Actions. As part of the penetration test, we implemented JUnit tests evaluating this weakness with 12 malicious files and created a pull request to sustainably prevent this kind of weakness in the development process. We also presented countermeasures circumventing the identified weaknesses and recommend implementing them.

Structure. The report is structured as follows: In Section 2, the timeline of the penetration test is listed. Section 3 introduces our methodology and Section 4 explains the general conditions and scope of the penetration test. Section 5 provides an overview of the identified weaknesses, as well as, further recommendations and information. In Section 6, all identified weaknesses are discussed in detail and specific countermeasures are described. Section 7 summarizes our recommendations resulting from observations of the application. Finally, Section 8 lists additional tests that did not reveal any weaknesses.

2 Project Timeline

The source code analysis and penetration test were carried out remotely from 11.12.2023 to 19.12.2023. The library "openconext-saml-java" developed by SURF, as well as, an IdP test implementation provided by SURF were examined by two people with the entire effort of four man-days – including documentation and writing of this report.

3 Methodology

Among others, the following tools were used for the penetration test:

Tool	Link
Mozilla Firefox	https://www.mozilla.org/de/firefox/
Burp Suite Professional	https://portswigger.net/burp
Self-developed tools	-

Risk Rating. Each weakness has its own CVSS 3.1 base score rating (*Common Vulnerability Scoring System Version 3.1 Calculator*).[1,2] Based on the CVSS 3.1 base score, the following weaknesses assessment is performed:

0.0 – 3.9:	Low
4.0 – 6.9:	Medium
7.0 – 8.9:	High
9.0 – 10.0:	Critical

[1] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator
[2] https://www.first.org/cvss/v3.1/user-guide

4 General Conditions and Scope

In the scope of the penetration test / source code analysis was the open-source library "openconext-saml-java" developed by SURF, as well as, a test implementation provided by SURF. The "openconext-saml-java" library allows to implement a SAML-based identity provider (IdP). The source code is available on GitHub[3] and was examined at the stage of commit `86f31be28d04d4da65a39b193591959536d9f752`.

SURF provided access to a test environment with a running IdP called "eduID (NL) test environment" and SAML service provider (SP) called "Open ID Connect Playground"[4]. The SP was not in the scope of the penetration test. Hackmanit could register own test accounts at the test IdP for the penetration test. The test services were available at the following URLs:

Service	URL
IdP	https://login.test.eduid.nl
SP (out of scope)	https://oidc-playground.test.surfconext.nl

Table 1: Overview of test services used during the penetration test.

[3] https://github.com/OpenConext/openconext-saml-java
[4] Despite its name the SP supported both OpenID Connect (OIDC) and SAML.

5 Overview of Weaknesses, Recommendations, and Information

Risk Level	Finding	Reference
M01	**XML Signature Wrapping Attack Targeting Signed Authentication Requests:** The IdP is vulnerable to XML Signature Wrapping attacks. This allows an attacker to bypass the protection provided by the signature and arbitrarily manipulate the request's contents.	Section 6.1, page 7
R01	**Improving the Configuration of Security-Related Headers:** The configuration of security-related HTTP headers set by the IdP should be improved to further harden its security.	Section 7.1, page 11

Risk Definitions:

Critical Risk	Weaknesses classified as *Critical* can be exploited with very little effort by an attacker. They have very large negative effects on the tested system, its users and data, or the system environment.
High Risk	Weaknesses classified as *High* can be exploited with little effort by an attacker. They have a major negative impact on the tested system, its users and data, or the system environment.
Medium Risk	Weaknesses classified as *Medium* can be exploited with medium effort by an attacker. They have a medium negative impact on the tested system, its users and data, or the system environment.
Low Risk	Weaknesses classified as *Low* can only be exploited with great effort by an attacker. They have little negative impact on the tested system, its users and data, or the system environment.
Recommendation	*Recommendation* identifies measures that may increase the security of the tested system. Implementation is recommended, but not necessarily required.
Information	Observations classified as *Information* are usually no weaknesses. Examples of these observations are unusual configurations and possibly unwanted behavior of the tested system.

6 Weaknesses

In the following sections, we list the identified weaknesses. Every weakness has an identification name which can be used as a reference in the event of questions, or during the patching phase.

6.1 M01 XML Signature Wrapping Attack Targeting Signed Authentication Requests

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	**Network**	Confidentiality Impact (C)	**Low**
Attack Complexity (AC)	**Low**	Integrity Impact (I)	**Low**
Privileges Required (PR)	**Low**	Availability Impact (A)	**None**
User Interaction (UI)	**Required**	Scope (S)	**Unchanged**
Subscore: **2.1**		Subscore: **2.5**	

Overall CVSS Score for M01 : 4.6

General Description. To protect the integrity and authenticity of SAML authentication requests, the whole `AuthnRequest` element can be digitally signed. The SAML specification [1] prescribes that the `AuthnRequest` element is protected with an enveloped XML Signature [2].

Listing 1 provides an example of a signed SAML authentication request.

The present XML Signature references the `AuthnRequest` root element using its ID (`A524bd1eb-20b6-4f60-8d83-e56470b77580`) and protects its integrity by computing the digest value. This ensures authenticity and integrity of data contained in the `AuthnRequest` element, for example, the attributes `Destination` and `AssertionConsumerServiceURL`.

```
1  <saml2p:AuthnRequest xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol"
       AssertionConsumerServiceURL="https://engine.test.surfconext.nl/authentication/sp/
       consume-assertion" Destination="https://mujina-idp.test.surfconext.nl/
       SingleSignOnService" ForceAuthn="true" ID="A524bd1eb-20b6-4f60-8d83-e56470b77580"
       IsPassive="false" IssueInstant="2023-12-15T12:42:49.740Z" ProtocolBinding="urn:oasis
       :names:tc:SAML:2.0:bindings:HTTP-POST" Version="2.0">
2     <saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">https://engine.
          test.surfconext.nl/authentication/sp/metadata</saml2:Issuer>
3     <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
4        <ds:SignedInfo>
5           <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"
                />
6           <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-
                sha512"/>
7           <ds:Reference URI="#A524bd1eb-20b6-4f60-8d83-e56470b77580">
8              <ds:Transforms>
9                 <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
                      signature"/>
10                <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
11             </ds:Transforms>
12             <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha512"/>
13             <ds:DigestValue>...</ds:DigestValue>
14          </ds:Reference>
15       </ds:SignedInfo>
16       <ds:SignatureValue>
17          ...
18       </ds:SignatureValue>
19       <ds:KeyInfo>
20          ...
21       </ds:KeyInfo>
22    </ds:Signature>
23    <saml2p:RequestedAuthnContext>
24       <saml2:AuthnContextClassRef xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
             https://refeds.org/profile/mfa</saml2:AuthnContextClassRef>
25    </saml2p:RequestedAuthnContext>
26 </saml2p:AuthnRequest>
```

Listing 1: Example of a signed SAML authentication request.

For an IdP, it is critical to process only signed document parts [4]. For this purpose, the XML Signature verification and dereferencing of signed XML parts has to be performed; if an attacker is able to force the IdP to process unsigned data, they can arbitrarily modify the `AuthnRequest` element's contents.

Weakness. The source code analysis revealed that the tested IdP is vulnerable to XML signature wrapping attacks. A successful attack is depicted in Listing 2 and works as follows:

1. An attacker executing the attack moves the original signed content to a different element. In this case, the whole `AuthnRequest` element is copied into the `RequestedAuthnContext` element (lines 25 - 30). Since the content is not modified, the XML Signature can still be successfully validated.

2. The attacker then removes the ID of the `AuthnRequest` root element (line 1); this ensures that the `AuthnRequest` root element is not referenced by the signature and thus is not integrity protected.

3. The attacker can finally change the content of the `AuthnRequest` root element. In this example, the `Destination` URL is changed to `https://hackmanit.de/SingleSignOnService` (line 1).

```
1  <saml2p:AuthnRequest xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol"
       AssertionConsumerServiceURL="https://engine.test.surfconext.nl/authentication/sp/
       consume-assertion" Destination="https://hackmanit.de/SingleSignOnService" ForceAuthn
       ="true" IsPassive="false" IssueInstant="2023-12-15T12:42:49.740Z" ProtocolBinding="
       urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" Version="2.0">
2     <saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">https://engine.
          test.surfconext.nl/authentication/sp/metadata</saml2:Issuer>
3     <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
4        <ds:SignedInfo>
5           <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"
                />
6           <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-
                sha512"/>
7           <ds:Reference URI="#A524bd1eb-20b6-4f60-8d83-e56470b77580">
8              <ds:Transforms>
9                 <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
                      signature"/>
10                <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
11             </ds:Transforms>
12             <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha512"/>
13             <ds:DigestValue>...</ds:DigestValue>
14          </ds:Reference>
15       </ds:SignedInfo>
16       <ds:SignatureValue>
17          ...
18       </ds:SignatureValue>
19       <ds:KeyInfo>
20          ...
21       </ds:KeyInfo>
22    </ds:Signature>
23    <saml2p:RequestedAuthnContext>
24       <saml2:AuthnContextClassRef xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
             https://refeds.org/profile/mfa</saml2:AuthnContextClassRef>
25       <saml2p:AuthnRequest xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol"
             AssertionConsumerServiceURL="https://engine.test.surfconext.nl/authentication/
             sp/consume-assertion" Destination="https://mujina-idp.test.surfconext.nl/
             SingleSignOnService" ForceAuthn="true" ID="A524bd1eb-20b6-4f60-8d83-
             e56470b77580" IsPassive="false" IssueInstant="2023-12-15T12:42:49.740Z"
             ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" Version="2.0"
             >
26          <saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">https://
                engine.test.surfconext.nl/authentication/sp/metadata</saml2:Issuer>
27          <saml2p:RequestedAuthnContext>
28             <saml2:AuthnContextClassRef xmlns:saml2="urn:oasis:names:tc:SAML:2.0:
                   assertion">https://refeds.org/profile/mfa</saml2:AuthnContextClassRef>
29          </saml2p:RequestedAuthnContext>
30       </saml2p:AuthnRequest>
31    </saml2p:RequestedAuthnContext>
32 </saml2p:AuthnRequest>
```

Listing 2: Successful XML signature wrapping attack targeting a signed SAML authentication request.

The IdP processing such message can successfully validate the signature over the original `AuthnRequest` element (lines 25 - 30). However, it retrieves the data from the modified `AuthnRequest` root element (line 1) and thus processes `https://hackmanit.de/SingleSignOnService` as a valid `Destination`.

Note that, while the XML signature wrapping attack breaks the authenticity and integrity of the XML Signature, its impact on SAML authentication request messages is rather small; SAML authentication requests typically only contain data, which are defined in the IdP metadata. Changing values such as `AssertionConsumerServiceURL` can be thus detected by other

means. Nevertheless, relying on a vulnerable XML Signature validation could introduce unrestricted trust in the message contents and result in weaknesses in future deployments.

The evaluated implementation uses the `SignatureValidator.validate(signature, credential)` function of the OpenSAML library's validation logic. However, this function only verifies the cryptographic correctness of the XML Signature.

To prevent XML signature wrapping attacks, the implementation needs to ensure that the processed message contents have been verified in the signature validation step. For this purpose, OpenSAML offers the `SAMLSignatureProfileValidator` class with its `validate` function. `SAMLSignatureProfileValidator` evaluates whether the included signature protects the correct root element and executes further checks preventing XML signature wrapping attacks.

In the scope of the penetration test, we extended the JUnit tests of the openconext-saml-java library with 12 XML signature wrapping test cases to evaluate this class of attacks directly during the development.[5]

Countermeasures. We recommend using the `SAMLSignatureProfileValidator` class with its `validate` function provided by the OpenSAML library to validate XML Signatures.

[5] https://github.com/OpenConext/openconext-saml-java/pull/1

7 Recommendations

In the following sections, we provide our recommendations to improve the security of the tested system.

7.1 R01 Improving the Configuration of Security-Related Headers

General Description. There are some HTTP headers that instruct browsers to enable security mechanisms. These security mechanisms are used to protect against attacks such as clickjacking, cross-site scripting (XSS), or man-in-the-middle (MitM).

The IdP frontend uses the security-related headers `X-Frame-Options`, `Content-Security-Policy`, `X-Content-Type-Options`, `X-XSS-Protection`, and `Strict-Transport-Security`. An example of the HTTP headers set by the IdP is depicted in Listing 3.

```
HTTP/1.1 200
date: Fri, 15 Dec 2023 10:36:41 GMT
server: Apache
content-security-policy: default-src 'none'; script-src 'self' 'unsafe-inline'; style-
    src 'self' 'unsafe-inline'; font-src 'self'; connect-src 'self' https://connect.test
    .surfconext.nl; img-src 'self' https://static.surfconext.nl data:; form-action 'self
    ' https://*.test.surfconext.nl; frame-ancestors 'none'; base-uri 'none'
x-frame-options: DENY
referrer-policy: same-origin
x-content-type-options: nosniff
cache-control: no-cache, no-store, max-age=0, must-revalidate
pragma: no-cache
expires: 0
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
x-frame-options: DENY
content-type: text/html;charset=UTF-8
set-cookie: HTTPSERVERID=javaapps|ZXwsP; path=/; HttpOnly; Secure; SameSite=None
strict-transport-security: max-age=34214400
connection: close
Content-Length: 15888

[...]
```

Listing 3: Example of a response with the HTTP headers set by the IdP.

The configuration of the security-related HTTP headers can be improved by the following adjustments:

`Content-Security-Policy` The content security policy (CSP) used by the IdP contains the value `unsafe-inline` for the directives `script-src` and `style-src`. This prevents the CSP from being an effective countermeasure against attacks such as XSS. The value `unsafe-inline` should be removed if possible. An example CSP including more detailed explanations can be found on the following page: https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html#csp-sample-policies.

`X-XSS-Protection` The HTTP header `X-XSS-Protection` can be used to configure the state of the XSS auditor (Google Chrome) or XSS filter (Internet Explorer). In the

past, it was recommended to deliver the HTTP header `X-XSS-Protection: 1; mode=block` so that XSS attacks can be blocked in case of detection via an error page. However, due to cross-site leak (XS-Leak) attacks, it is recommended to disable the XSS protection mechanism. Browser vendors such as Google no longer deliver a filter or auditor for protection against XSS in new versions.[6] We therefore recommend explicitly setting the HTTP header to `X-XSS-Protection: 0` to disable the XSS auditor or XSS filter and thus prevent any possible XS-Leak attacks.

`Strict-Transport-Security` To extend the protection against MitM attacks, the value of this header could be extended to `Strict-Transport-Security: max-age=34214400; includeSubDomains`.

`Server` To prevent disclosing information about the software in use, we recommend removing the `Server` header. This information could be valuable to attackers and be used for further attacks.

Duplicate Headers The two headers `X-Content-Type-Options` and `X-Frame-Options` are present twice in the IdP's responses. It is likely that the duplicate headers are set by different systems or applications. The duplicate headers have the same value in the tested case. However, it is possible that different systems use different values for the headers. This could result in unintended behavior on the receiver side [3]. We recommend making sure each header is only set by one system.

Recommendation. We recommend adjusting the HTTP header configuration of the IdP as described above.

[6] https://www.chromium.org/developers/design-documents/xss-auditor

8 Further Evaluations

In this section, we list further evaluations we conducted in our penetration test. It provides useful information for future security evaluations.

XML Parser Configuration. The library defines the configuration of the XML parser in the `getParserBuilderFeatures()` function (lines 156-165 in `DefaultSAMLService.java`). The configuration was examined for missing security-related settings. No weaknesses could be identified. Most importantly the processing of Document Type Definitions (DTDs) is disabled.

Document Type Definition (DTD) / XML External Entity (XXE). As described above the processing of DTDs is disabled according to the XML parser configuration. It was practically evaluated whether this configuration is effective or if DTDs are processed anyway. During the penetration test it was not possible to make the IdP process DTDs. The tested payloads for XXE attacks were not successful. The IdP responded with an error message in all tested cases.

XML Injection. It was evaluated whether it was possible to inject XML elements or attributes into the SAML response or SAML assertion by manipulating elements or attributes in the SAML request. For example, `quot; hackmanit= quot;XYZ` was added at the end of the `ID` attribute of the SAML request. The IdP did not decode the XML entities in the injected string and used the exact same value of the `ID` attribute as the value of the `InResponseTo` attribute in the SAML response.

ACS Spoofing. The IdP was evaluated for ACS spoofing attacks. Their goal is to make the IdP sent the SAML assertion to an attacker-controlled domain instead of the SP. For this purpose, the attribute `AssertionConsumerServiceURL` present in the SAML request was replaced with different manipulated values. The IdP was not vulnerable to ACS spoofing attacks. Manipulating the domain or path of the genuine `AssertionConsumerServiceURL` of the SP resulted in the IdP rejecting the SAML request with an error message. The same happened when the whole URL was replaced with an arbitrary one (e.g., `https://hackmanit.de`), an invalid value (e.g., `null`), or when the attribute was removed completely.

Server-Side Request Forgery (SSRF). The SAML request contains URLs in multiple elements and attributes. When these URLs were replaced with an URL controlled by Hackmanit the IdP did not invoke the URL. It was not possible to make the IdP access arbitrary URLs.

Manipulations of the `RequesterID` Elements. The `RequesterID` elements in the SAML request were altered or removed. The presence and value of a `RequesterID` element influences the `AttributeStatement` in the SAML response. While most `Attribute` elements containing information about the authenticated user are not influences by the `RequesterID` elements in the SAML request the attribute called `urn:mace:eduid.nl:1.1` contains different values when the `RequesterID` element in the SAML request differs. According to SURF this is intended behavior.

Other Manipulations in the SAML Request. The SAML request was manipulated in other ways such as manipulating the `Issuer` element, the `Destination` or `ProtocolBinding` attributes, or adding a `NameIDPolicy` element or `IsPassive` attribute. None of these manipulations revealed any weaknesses. In addition, adding a `Subject` element with a user

identity other than the account used for the subsequent login, did not have any effect on the issued SAML assertion. The assertion contained the correct identifiers for the identity of the account logged in during the protocol flow. Except for the `Issuer` element, the elements and attributes mentioned above seem to be ignored by the IdP.

RelayState. A new `RelayState` parameter was added to the request and sent along with the `SAMLRequest`. The `RelayState` was processed and reflected by the IdP. However, it was not possible to find an exploitable reflection leading to weaknesses.

Node Splitting Attacks. The IdP was examined for its vulnerability to node splitting attacks, which exploit splitting of validated text nodes after successful XML Signature validation.[7] Different variants of these attacks with the help of comments and CDATA sections were tested. None of the tests was successful.

[7] https://i.blackhat.com/us-18/Thu-August-9/us-18-Ludwig-Identity-Theft-Attacks-On-SSO-Systems.pdf

9 References

[1] Scott Cantor et al. *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0*. OASIS Standard. Organization for the Advancement of Structured Information Standards, 2005. url: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

[2] Donald Eastlake et al. "XML Signature Syntax and Processing (Second Edition)". In: *W3C Recommendation* (2008).

[3] Hendrik Siewert et al. "On the Security of Parsing Security-Relevant HTTP Headers in Modern Browsers". In: *2022 IEEE Security and Privacy Workshops (SPW)*. 2022, pp. 342–352. doi: 10.1109/SPW54247.2022.9833880.

[4] Juraj Somorovsky et al. "On Breaking SAML: Be Whoever You Want to Be". In: *21st USENIX Security Symposium*. Bellevue, WA, Aug. 2012.