Penetration Test Report:

WAYF Identity Provider (SAML and OpenID Connect)

Version: 1.0.2
12.12.2023

Karsten Meyer zu Selhausen
Phone: +49(0)234 / 54459996 | E-Mail: karsten.meyerzuselhausen@hackmanit.de

Project Information

Customer:	WAYF / Danish e-Infrastructure Cooperation Asmussens Allé, Building 305 DK-2800 Lyngby, DENMARK
Contact:	Mads Freek Petersen
Commissioned to:	Hackmanit GmbH Universitätsstraße 60 (Exzenterhaus) 44789 Bochum, Germany
Project executive:	Karsten Meyer zu Selhausen Phone: +49(0)234 / 54459996 Fax: +49(0)234 / 54427593 E-Mail: karsten.meyerzuselhausen@hackmanit.de
Project members:	Maximilian Hildebrand (Hackmanit GmbH) Sebastian Krause (Hackmanit GmbH) Prof. Dr. Juraj Somorovsky (Hackmanit GmbH)
Project period:	2023-04-11 – 2023-05-09
Version of the report:	1.0.2

This report was technically verified by Karsten Meyer zu Selhausen.
This report was linguistically verified by Conrad Schmidt.

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 WAYF to perform a penetration test of their identity provider (IdP) implementation (referred to as IdP_W in this report). IdP_W serves as an intermediary in the WAYF ecosystem between service providers (SPs) and IdPs operated by different parties. The penetration test was performed remotely with a total expense of 10 man-days – including documentation and writing of this report.

Weaknesses. During the penetration test a total of three weaknesses were identified: one weakness classified as *Critical* and two weaknesses classified as *Medium*.

The highest ranked weakness, C01, allows an attacker to impersonate any victim registered at any IdP in the WAYF ecosystem when authenticating to IdP_W. Afterwards, IdP_W issues a response to the SP containing the victim's identity. This allows the attacker to impersonate any user at any SP in the WAYF ecosystem. IdP_W is vulnerable because it allows to initiate an authentication flow with an arbitrary IdP but to finish the flow with a response of an IdP controlled by the attacker.

The first weakness classified as *Medium*, M01, allows to embed the consent-page and other pages of IdP_W into an iframe. This can be used by an attacker to trick a victim into unintentionally performing a login flow.

The second weakness classified as *Medium*, M02, enables an attacker to use ID tokens or SAML assertions – issued for a particular party but without valid "audience" information – for another party. This can ultimately enable the attacker to take over the victim's account.

Top Weaknesses:

Risk Level	Finding	Reference
C01	ID Spoofing – Impersonation of Arbitrary Victims Possible	Section 6.1, page 14

Recommended Actions. We recommend implementing the countermeasures described for the three identified weaknesses. Especially the countermeasures for C01 should be thoroughly implemented to mitigate the hijacking of arbitrary accounts without user interaction. It should also be evaluated to implement the four recommendations.

We additionally recommend to perform a retest to ensure that the proposed countermeasures are effective and the risk is successfully mitigated.

Retest. WAYF followed our recommendations and implemented the proposed countermeasures. We conducted a retest and can confirm that all identified weaknesses were successfully fixed. WAYF implemented three out of our four additional recommendations and resolved both information.

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. In Section 8, observations of unusual configurations and possibly unwanted behavior of the application are described. Finally, Section 9 lists additional tests that did not reveal any weaknesses.

2 Project Timeline

The penetration test was carried out remotely from 2023-04-11 to 2023-05-09. The WAYF implementation of their IdP (referred to as IdP_W in this report) was examined by four people with the entire effort of 10 man-days – including documentation and writing of this report.

Some final testing was done from 2023-04-24 to 2023-04-26 as agreed.

While this penetration test reports was written a new attack idea emerged and was evaluated on 2023-05-08 and 2023-05-09. These tests resulted in identification of C01 .

Hackmanit conducted a retest of the identified weaknesses later in 2023 and can confirm that all weaknesses were successfully fixed. In addition three out of the four recommendations listed in Section 7 were fulfilled and both information listed in Section 8 were resolved, as well.

3 Methodology

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

Tool	Link
Mozilla Firefox	https://www.mozilla.org/de/firefox/
Google Chrome	https://www.google.com/intl/de_ALL/chrome/
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

4 General Conditions and Scope

In the scope of the penetration test was the implementation of WAYF's IdP (referred to as IdP_W in this report). IdP_W is based on "wayfhybrid" which is available on GitHub: https://github.com/wayf-dk/wayfhybrid.

WAYF provided access to a test environment including an IdP running the "wayhybrid" implementation, a SP supporting both Security Assertion Markup Language (SAML) and OpenID

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

Connect (OIDC), and accounts at a real Identity Provider (GrandUnified) (IdP_G). WAYF also added our own Identity Provider (Hackmanit) (IdP_H) to the ecosystem. The test services were available at the following URLs:

Service	URL
IdP_W	https://wayf.wayf.dk
SP	https://wayfsp.wayf.dk
IdP_G	https://grandunified.deic.dk

Table 2: Overview of test services used during the penetration test.

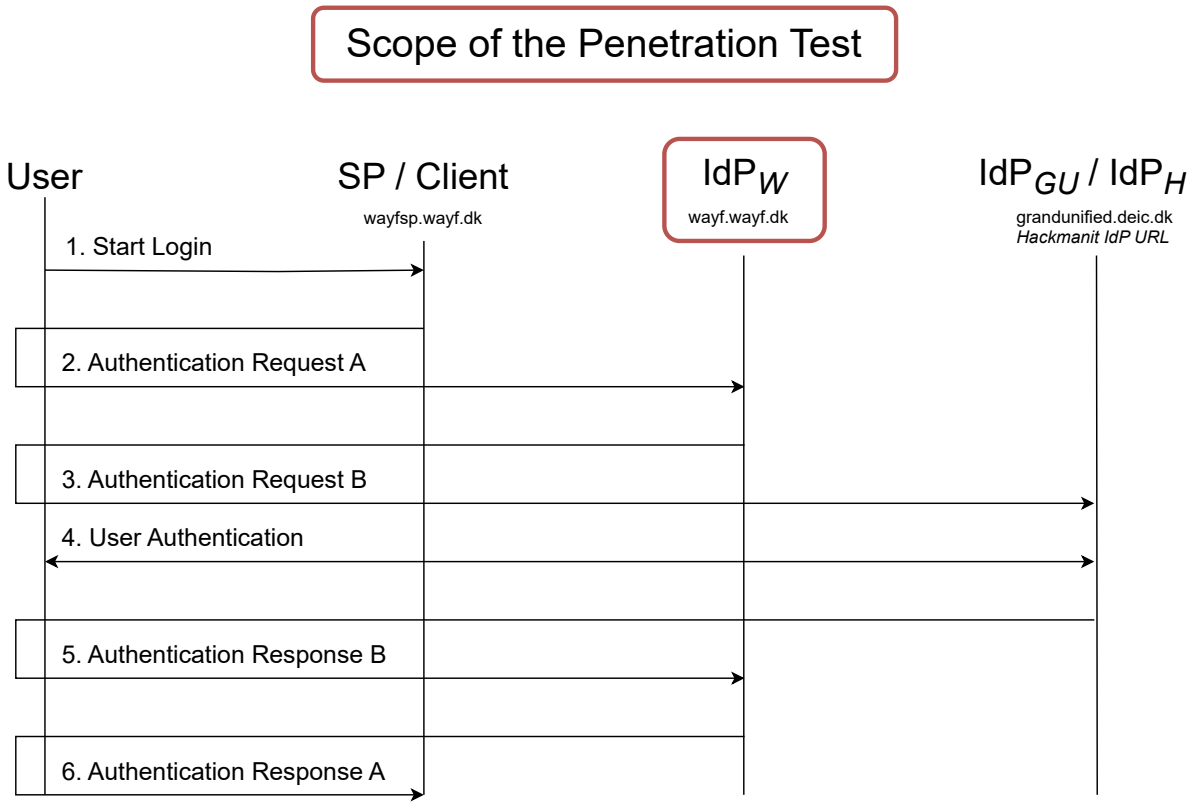

Figure 1: A high level overview of the tested SAML and OIDC protocol flow.

The deployed scenario is depicted in Figure 1 and contains of the following steps:

Step 1: The user visits the SP and starts the login process.

Step 2: A SAML request is generated and the user is redirected to IdP_W. An example for the SAML request look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<samlp:AuthnRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:saml=
    "urn:oasis:names:tc:SAML:2.0:assertion" Version="2.0" ProtocolBinding="urn:
    oasis:names:tc:SAML:2.0:bindings:HTTP-POST" ID="_zfoSr31Gd4Dial25akOYwa49Jawr"
     IssueInstant="2023-04-11T10:39:27Z" Destination="https://wayf.wayf.dk/saml2/
    idp/SSOService2.php" AssertionConsumerServiceURL="https://wayfsp.wayf.dk/ACS">
   <saml:Issuer>https://wayfsp.wayf.dk</saml:Issuer>
   <samlp:NameIDPolicy Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient
       " AllowCreate="true"/>
</samlp:AuthnRequest>
```

Listing 1: SAML request in Step 2.

Step 3: The user selects an IdP (e.g., IdP_G). IdP_W generates a SAML request and the user is forwarded to IdP_G:

```
<samlp:AuthnRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:saml=
    "urn:oasis:names:tc:SAML:2.0:assertion" Version="2.0" ID="
    _ywZyEgXQEScrynHKxTrQhjvewX9U" IssueInstant="2023-04-11T10:38:03Z" Destination
    ="https://grandunified.deic.dk/SSO" ProtocolBinding="urn:oasis:names:tc:SAML
    :2.0:bindings:HTTP-POST" AssertionConsumerServiceURL="https://wayf.wayf.dk/
    module.php/saml/sp/saml2-acs.php/wayf.wayf.dk" ProviderName="WAYF Testing
    Service">
   <saml:Issuer>https://wayf.wayf.dk</saml:Issuer>
   <samlp:NameIDPolicy Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient
       " AllowCreate="true"/>
   <samlp:Scoping>
      <samlp:RequesterID>https://wayfsp.wayf.dk</samlp:RequesterID>
   </samlp:Scoping>
</samlp:AuthnRequest>
```

Listing 2: SAML request in Step 3.

Step 4: If the user is not logged in at IdP_G, they are requested to authenticate.

Step 5: After a successful authentication IdP_G issues a SAML response which is sent to IdP_W. Its content is shown in Listing 3. The SAML response contains a signed SAML assertion. The `<Response>` element is not protected.

```
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:saml="urn
    :oasis:names:tc:SAML:2.0:assertion" xmlns:xs="http://www.w3.org/2001/XMLSchema
    " Version="2.0" ID="_x81U8EdqcMlN3DAc2eF28EYsnAWl" IssueInstant="2023-04-11T10
    :38:04Z" InResponseTo="_ywZyEgXQEScrynHKxTrQhjvewX9U" Destination="https://
    wayf.wayf.dk/module.php/saml/sp/saml2-acs.php/wayf.wayf.dk">
<saml:Issuer>https://guidp.deic.dk</saml:Issuer>
<samlp:Status>
   <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
</samlp:Status>
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" Version="2.0"
    ID="_A1u0Z6wKkAQl5gNdITly6aG5V32I" IssueInstant="2023-04-11T10:38:04Z">
   <saml:Issuer>https://guidp.deic.dk</saml:Issuer>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
[...]
   </ds:Signature>
   <saml:Subject>
   <saml:NameID SPNameQualifier="https://wayf.wayf.dk" Format="urn:oasis:names:tc:
       SAML:2.0:nameid-format:transient">_VKXIRbmIXjA2FTTjciDe27XRERj-</saml:
       NameID>
   <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
      <saml:SubjectConfirmationData NotOnOrAfter="2023-04-11T10:42:04Z" Recipient=
          "https://wayf.wayf.dk/module.php/saml/sp/saml2-acs.php/wayf.wayf.dk"
          InResponseTo="_ywZyEgXQEScrynHKxTrQhjvewX9U"/>
   </saml:SubjectConfirmation>
   </saml:Subject>
   <saml:Conditions NotBefore="2023-04-11T10:38:04Z" NotOnOrAfter="2023-04-11T10
       :42:04Z">
   <saml:AudienceRestriction>
      <saml:Audience>https://wayf.wayf.dk</saml:Audience>
   </saml:AudienceRestriction>
   </saml:Conditions>
[...]
   <saml:Attribute Name="cn" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
       format:basic">
      <saml:AttributeValue>Maximilian Hildebrand</saml:AttributeValue>
   </saml:Attribute>
   <saml:Attribute Name="eduPersonPrincipalName" NameFormat="urn:oasis:names:tc:
       SAML:2.0:attrname-format:basic">
      <saml:AttributeValue>qear-keag-bean-koat-so@grandunified.deic.dk</saml:
          AttributeValue>
   </saml:Attribute>
   <saml:Attribute Name="mail" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
       format:basic">
      <saml:AttributeValue>maximilian.hildebrand@hackmanit.de</saml:AttributeValue
          >
   </saml:Attribute>
   </saml:AttributeStatement>
</saml:Assertion>
</samlp:Response>
```

Listing 3: SAML response in Step 5 (*condensed*).

Step 6: IdP_W validates the SAML assertion and generates a SAML response for the SP. Finally, the user is redirected back to the SP with the SAML response shown in Listing 4.

```
1   <samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:saml="
        urn:oasis:names:tc:SAML:2.0:assertion" xmlns:xs="http://www.w3.org/2001/
        XMLSchema" Version="2.0" ID="_trFdJStW1f6zfnDwFHIgTqeYV5N4" IssueInstant="
        2023-04-11T10:38:04Z" InResponseTo="_zfoSr31Gd4Dial25akOYwa49Jawr"
        Destination="https://wayfsp.wayf.dk/ACS">
2   <saml:Issuer>https://wayf.wayf.dk</saml:Issuer>
3   <samlp:Status>
4     <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
5   </samlp:Status>
6   <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" Version="2.0
        " ID="_n5sH8kIvtFwl_8-BSozpv3frJj48" IssueInstant="2023-04-11T10:38:04Z">
7     <saml:Issuer>https://wayf.wayf.dk</saml:Issuer>
8     <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
9 [...]
10    </ds:Signature>
11    <saml:Subject>
12      <saml:NameID SPNameQualifier="https://wayfsp.wayf.dk" Format="urn:oasis:
            names:tc:SAML:2.0:nameid-format:transient">_Le98uTczFt2ZtjOPx7JByY-FIJY3
            </saml:NameID>
13      <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
14        <saml:SubjectConfirmationData NotOnOrAfter="2023-04-11T10:42:04Z" Recipient
              ="https://wayfsp.wayf.dk/ACS" InResponseTo="
              _zfoSr31Gd4Dial25akOYwa49Jawr"/>
15      </saml:SubjectConfirmation>
16    </saml:Subject>
17    <saml:Conditions NotBefore="2023-04-11T10:38:04Z" NotOnOrAfter="2023-04-11T11
          :39:04Z">
18      <saml:AudienceRestriction>
19        <saml:Audience>https://wayfsp.wayf.dk</saml:Audience>
20      </saml:AudienceRestriction>
21    </saml:Conditions>
22    <saml:AuthnStatement AuthnInstant="2023-04-11T10:38:04Z" SessionIndex="
          _flu1ayWBnmyqjerSShgq47jE8Uf1" SessionNotOnOrAfter="2023-04-11T14:38:04Z">
23      <saml:AuthnContext>
24        <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:
              PasswordProtectedTransport</saml:AuthnContextClassRef>
25        <saml:AuthenticatingAuthority>https://guidp.deic.dk</saml:
              AuthenticatingAuthority>
26      </saml:AuthnContext>
27    </saml:AuthnStatement>
28    <saml:AttributeStatement>
29      <saml:Attribute Name="cn" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
            format:basic" FriendlyName="cn">
30        <saml:AttributeValue>Maximilian Hildebrand</saml:AttributeValue>
31      </saml:Attribute>
32      <saml:Attribute Name="eduPersonPrincipalName" NameFormat="urn:oasis:names:tc
            :SAML:2.0:attrname-format:basic" FriendlyName="eduPersonPrincipalName">
33        <saml:AttributeValue>qear-keag-bean-koat-so@grandunified.deic.dk</saml:
              AttributeValue>
34      </saml:Attribute>
35 [...]
36    </saml:AttributeStatement>
37  </saml:Assertion>
38 </samlp:Response>
```

Listing 4: SAML response in Step 6 (*condensed*).

Alternatively, OIDC can be used between SP and IdP_W instead of SAML. In this case, steps 2 and 6 differ from the flow described above.

Step 2: The user is redirected to IdP_W. The request contains four parameters:

- `response_type=id_token`

- `redirect_url=https%3A%2F%2Fwayfsp.wayf.dk%2FACS`
- `client_id=https%3A%2F%2Fwayfsp.wayf.dk`
- `idpentityid=https%3A%2F%2Fguidp.deic.dk`

Step 6: IdP_W validates the SAML assertion and generates an ID token for the SP. Finally, the user is redirected back to the SP with the ID token. The ID token can be optionally encrypted depending on the configuration of the SP and IdP_W. An example of an unencrypted ID token is shown in Listing 5.

```
[...].
{
   "aud": "https://wayfsp.wayf.dk",
   "cn": [
      "Karsten Meyer zu Selhausen"
   ],
   "eduPersonPrincipalName": [
      "karsten.meyerzuselhausen@hackmanit.de"
   ],
   "exp": 1681902981,
   "iat": 1681899321,
   "iss": "https://wayf.wayf.dk",
   "nbf": 1681899321,
   "nonce": "",
   "sub": "karsten.meyerzuselhausen@hackmanit.de",
   [...]
}
.[...]
```

Listing 5: Decoded ID token in Step 6 *(condensed)*.

As a second alternative our own IdP, IdP_H, could be used instead of IdP_G. IdP_W supports the OIDC "Implicit Flow" (`response_type=id_token`) in addition to SAML. This allowed us to conduct extensive tests for both protocols using IdP_H. When using IdP_H with OIDC steps 3 and 5 differ from the flow described above. When using the OIDC implicit flow these steps look like this:

Step 3: The user selects an IdP (e.g., IdP_H). IdP_W forwards the user to IdP_H. An example authentication request for the implicit flow is depicted in Listing 6.

```
https://ec2-13-53-188-80.eu-north-1.compute.amazonaws.com/realms/IdP-OIDC/protocol
    /openid-connect/auth?audience=https%3A%2F%2Fwayf.wayf.dk&client_id=https%3A%2F
    %2Fwayf.wayf.dk&nonce=_Mjj8KeL4SGbGbdwKxPX0dLfNPvtb&redirect_uri=https%3A%2F%2
    Fwayf.wayf.dk%2Fmodule.php%2Fsaml%2Fsp%2Fsaml2-acs.php%2Fwayf.wayf.dk&
    response_mode=form_post&response_type=id_token&scope=openid&state=
```

Listing 6: OIDC authentication request in Step 3.

Step 5: After a successful authentication IdP_H issues an ID token response which is sent to IdP_W. An example of an ID token is shown in Listing 7.

```
[...].{
  "exp": 1682080181,
  "iat": 1682079281,
  "iss": "https://ec2-13-53-188-80.eu-north-1.compute.amazonaws.com/realms/IdP-OIDC
      ",
  "aud": "https://wayf.wayf.dk",
  "sub": "e2565a8c-f7e5-40ba-887f-52138c14ee15",
  "nonce": "_Mjj8KeL4SGbGbdwKxPX0dLfNPvtb",
  "cn": [
     "Karsten Meyer zu Selhausen"
  ],
  "eduPersonPrincipalName": [
     "karsten.meyerzuselhausen@hackmanit.de"
  ],
[...]
}.[...]
```

Listing 7: Decoded ID token in Step 6 (*condensed*).

5 Overview of Weaknesses, Recommendations, and Information

Risk Level	Finding	Reference
C01	**ID Spoofing – Impersonation of Arbitrary Victims Possible:** An attacker can use their own IdP to impersonate any user of any IdP and authenticate at IdP_W.	Section 6.1, page 14
M01	**Clickjacking Attack On IdP_W:** The frontend does not implement countermeasures against clickjacking attacks. This allows an attacker to trick a victim into logging in unintentionally.	Section 6.2, page 18
M02	**Insufficient Validation of "Audience" Information:** IdP_W validates both ID tokens and SAML assertions on reception. However, the validation of the "audience" information is not sufficient in all cases.	Section 6.3, page 19
R01	**Adding Security-Related Headers:** Additional security-related HTTP headers should be used to instruct browsers to enable security mechanisms that can prevent attacks.	Section 7.1, page 22
R02	**Improve the Replay Attack Protection:** Additional security parameters of a SAML response should be validated.	Section 7.2, page 24
R03	**Wrong Conversion of XML/JWT Encryption Algorithms:** The conversion from XML Encryption algorithms to JSON Web Algorithms uses wrong values.	Section 7.3, page 25
R04	**Insufficient Validation of `<Signature>` Elements' Count:** The "goxml" library uses an XPath expression that does not sufficiently restrict the number of `<Signature>` elements.	Section 7.4, page 26
I01	**Automatic Modification of the `eduPersonPrincipalName` Claim:** The `eduPersonPrincipalName` claim is extracted from an ID token and automatically modified by IdP_W.	Section 8.1, page 27

I02	**Improvable Handling of Unsupported `response_type` Values:** IdP_W does not reject unsupported values for the `response_type` parameter but issues SAML responses instead.	Section 8.2, page 28

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 C01 ID Spoofing – Impersonation of Arbitrary Victims Possible

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	**Network**	Confidentiality Impact (C)	**High**
Attack Complexity (AC)	**High**	Integrity Impact (I)	**High**
Privileges Required (PR)	**None**	Availability Impact (A)	**High**
User Interaction (UI)	**None**	Scope (S)	**Changed**
Subscore: **2.2**		Subscore: **6.0**	

Overall CVSS Score for C01 : 9.0

General Description. IdPs are only allowed to issue authentication tokens (e.g., OIDC ID tokens or SAML assertions) for identities in a specific realm (“scope” *called here*). For example, IdP A is only allowed to issue tokens about identities in its realm `@a.com`. IdP B is not allowed to issue tokens in realm `@a.com`.

The goal of an ID spoofing attack is to impersonate a victim and authenticate with the victim’s identity at a vulnerable service. A usual prerequisite of ID spoofing attacks is that the attacker is in control of an IdP which is trusted by the vulnerable service. The attacker uses their IdP to issue malicious authentication tokens. These tokens contain the identity of a victim whose identity is in the realm of another IdP. The attacker’s IdP is not allowed to issue tokens about identities in this realm. However, the vulnerable service does not validate which realm the IdP is allowed to issue tokens for. The service accepts the malicious authentication token and the attacker is authenticated as the victim.

Weakness. When a login flow is started IdP_W allows the user to select an IdP to authenticate. IdP_W stores data about the scope of the selected IdP, as well as, some data about the started protocol flow (e.g., `ID` of SAML request or value of `nonce` parameter in OIDC). Upon receiving the response and authentication token IdP_W verifies the signature of the authentication token and some data in this token. For example, IdP_W validates that the value used for the `ID` of SAML request or `nonce` parameter is present in the authentication token. IdP_W also validates that the identity contained in the authentication token is in the expected scope and whether the authentication token was issued by a trusted IdP. However, IdP_W does not check whether the token was issued by the same IdP that was selected at the beginning of this login flow and to which IdP_W sent the initial authentication request.

This missing check allows an attacker to execute the following ID spoofing attack:[3]

1. The attacker starts a login flow at the SP and is redirected to IdP_W.
2. The attacker selects the IdP which the victim is registered at. The protocol used depends on the selected IdP. For example: IdP_G which uses SAML.
3. The attacker uses their own IdP_H to create a valid SAML assertion with the following data (see Listing 8):
   - `InResponseTo` fields set to the `ID` of the SAML request issued by IdP_W in step 2.
   - `eduPersonPrincipalName` attribute set to the identity of the victim. The identity is in the scope of IdP_G. For example: `VICTIM@grandunified.deic.dk`
4. The attacker sends the malicious assertion to IdP_W using the same browser they started the flow with in step 1. This means the cookies set by IdP_W are sent along with the assertion.
5. IdP_W displays a consent-page. On the consent-page the logo and name of IdP_G is displayed along with the identity of the victim (see Figure 2).
6. The attacker confirms the consent-page.
7. IdP_W issues a SAML assertion to the SP. This assertion contains IdP_H as `AuthenticatingAuthority` but the identity of the victim in the scope of a different IdP (see Listing 9).
8. The attacker is successfully authenticated as the victim at the SP.

```
<saml:Assertion ID="ID_2fadb5ad-b263-4015-84bd-e8e8d283043c" IssueInstant="2023-05-09
    T08:36:08.451Z" Version="2.0"
xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
<saml:Issuer>https://ec2-13-53-188-80.eu-north-1.compute.amazonaws.com/realms/IdP</saml
    :Issuer>
[...]
      <saml:Audience>https://wayf.wayf.dk</saml:Audience>
[...]
   <saml:Attribute FriendlyName="eduPersonPrincipalName" Name="eduPersonPrincipalName"
       NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
      <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">
            VICTIM@grandunified.deic.dk</saml:AttributeValue>
   </saml:Attribute>
[...]
</saml:Assertion>
```

Listing 8: SAML assertion issued by IdP_H containing the victim's identity in a scope of another IdP *(condensed)*.

[3] Prerequisite: The attacker must be in control of an IdP which is trusted by IdP_W. We use IdP_H as an example here.

```
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" Version="2.0" ID="
    _YldeQa-C02aFcBsgLXDURg2QXUCd" IssueInstant="2023-05-09T08:37:27Z">
<saml:Issuer>https://wayf.wayf.dk</saml:Issuer>
[...]
      <saml:Audience>https://wayfsp.wayf.dk</saml:Audience>
[...]
      <saml:AuthenticatingAuthority>https://ec2-13-53-188-80.eu-north-1.compute.
          amazonaws.com/realms/IdP</saml:AuthenticatingAuthority>
[...]
   <saml:Attribute Name="eduPersonPrincipalName" NameFormat="urn:oasis:names:tc:SAML
       :2.0:attrname-format:basic" FriendlyName="eduPersonPrincipalName">
      <saml:AttributeValue>VICTIM@grandunified.deic.dk</saml:AttributeValue>
   </saml:Attribute>
[...]
</saml:Assertion>
```

Listing 9: SAML assertion issued by IdP_W containing the victim's identity in a scope of an IdP different to the one in `AuthenticatingAuthority` *(condensed)*.

Countermeasures. We recommend implementing the following checks when receiving authentication tokens (e.g., OIDC ID tokens or SAML assertions):

- Validate that the authentication token received was issued by the same IdP that was selected at the beginning of the flow and the authentication request was sent to.
- Validate that the issuer of an authentication token is allowed to issue tokens for the scope of the identity which is contained in the token.
- Validate that the authentication response was sent using the same protocol that was used to send the authentication request. For example, when IdP_W started a SAML flow only a SAML assertion should be accepted and no ID token.

IdP_W must reject authentication tokens if any of these checks fail.

During the penetration test IdP_G only supported SAML and no other IdP supporting OIDC was available. Therefore, similar ID spoofing attacks could not be tested for OIDC and ID tokens. Due to the shared code between IdP_W implementation's of OIDC and SAML it is reasonable to assume that IdP_W is vulnerable to ID spoofing attacks for OIDC, as well.

Retest. We can confirm that this weakness was successfully fixed. IdP_W conducts the checks recommended above and the attack described above is now detected:

IdP_W responds with an error messages in step 5 and aborts the authentication flow if any of the checks fails. If the authentication token received is issued by an IdP other than the one selected at the beginning of the flow the error `IdP mismatch` is raised. If IdP_W receives an authentication token from another flow than expected (e.g. an ID token instead of a SAML assertion, or the other way around) the error `protocol mismatch` is raised. If the issuer of the authentication token is not allowed to issue tokens for the scope of the identity contained in the token the error `["cause:security domain 'grandunified.deic.dk' does not match any scopes"]` is raised.

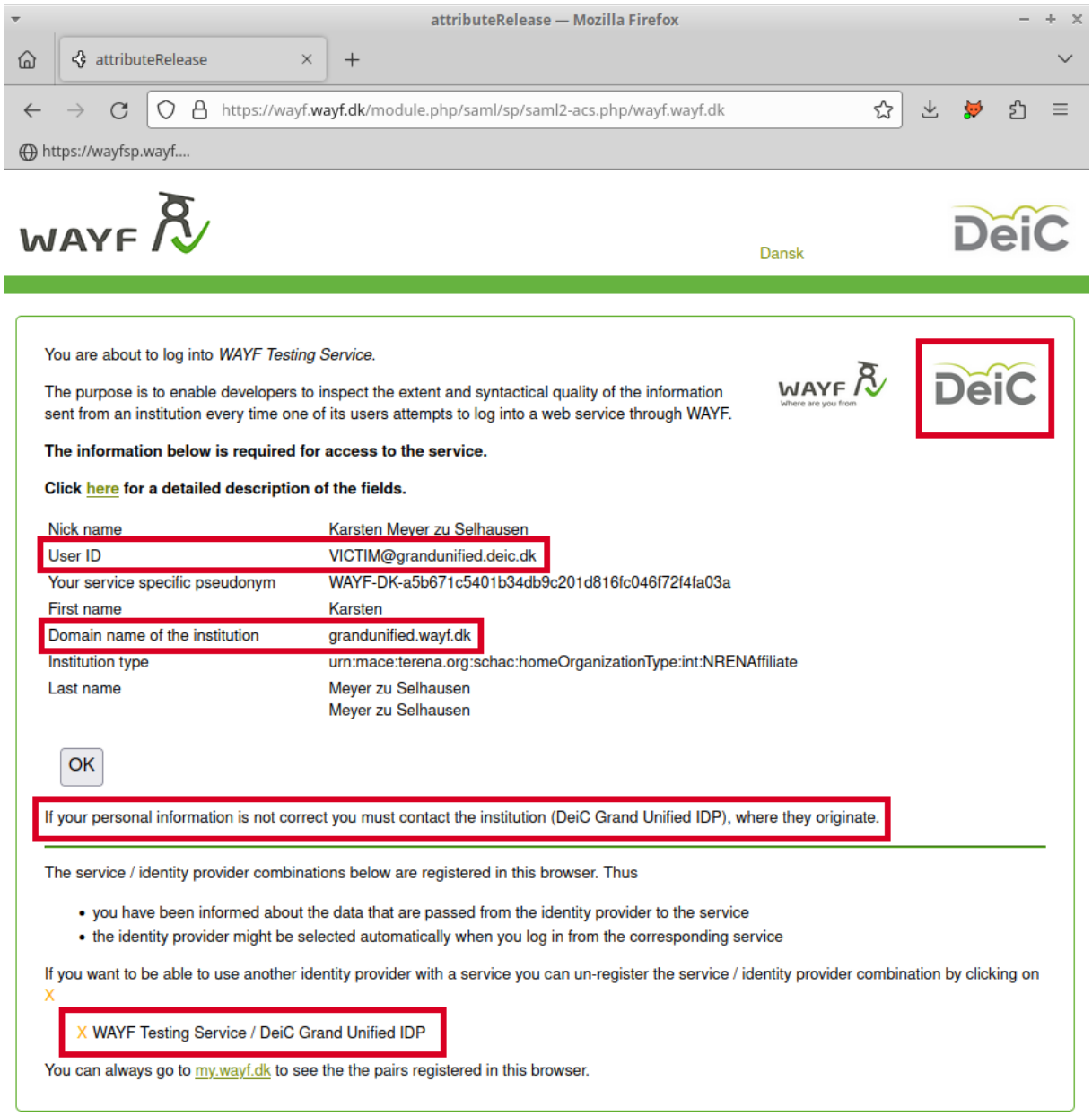

Figure 2: Consent-page displayed by IdP_W after receiving the SAML assertion in Listing 8 issued by IdP_H. The consent-page shows the logo and name of IdP_G and the identity of the victim in the scope of IdP_G.

6.2 M01 Clickjacking Attack On IdP$_W$

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	**Network**	Confidentiality Impact (C)	**Low**
Attack Complexity (AC)	**High**	Integrity Impact (I)	**Low**
Privileges Required (PR)	**None**	Availability Impact (A)	**None**
User Interaction (UI)	**Required**	Scope (S)	**Changed**
Subscore: **1.6**		Subscore: **2.7**	

Overall CVSS Score for M01 : 4.7

General Description. `Clickjacking`[4] is an attack in which an attacker attempts to trick a victim into performing unwanted actions in a Web application, usually while being logged in. The attacker can do this by creating a malicious Web page and luring the victim to visit that Web page. The vulnerable Web application is included in the malicious Web page as a transparent `iframe` – and thus invisible to the victim. For example, buttons are placed below the transparent `iframe` to entice the victim to click on them. When the victim tries to click on the visible buttons, the click is instead performed on the overlaying transparent `iframe` and triggers unwanted actions in the vulnerable Web application. The victim is unaware that these actions are being performed.

Weakness. The IdP$_W$ frontend can be loaded in an `iframe` and does not implement effective countermeasures against clickjacking attacks. The same is true for the `WAYF test service` SP, which was not in the scope of the penetration test. During the penetration test it was possible to embed the frontend of the `WAYF test service` as a transparent `iframe` into another website. Buttons were then placed in such a way that a victim would be tricked into pressing them in the correct order to successfully complete a login flow in the transparent `iframe` (see Figure 3). The first button starts the login flow at the SP, while the second button confirms the consent-page. Using this approach, an attacker could trick a victim into unintentionally logging into a SP. The only prerequisite is that the victim is already authenticated at the IdP that is used by IdP$_W$ to authenticate the user.

Countermeasures. We recommend providing both the HTTP header `X-Frame-Options` with the value `DENY` and the HTTP header `Content-Security-Policy` with the directive `frame-ancestors 'none'` in responses by IdP$_W$ and the SP.[5] This prevents clickjacking attacks in both older and modern browsers. R01 lists these and other HTTP header that we recommend to use.

Retest. We can confirm that this weakness was successfully fixed. Both IdP$_W$ and SP return the HTTP headers `Content-Security-Policy: frame-ancestors 'self'` and `X-Frame-Options: sameorigin`. This prevents them from being included in an `iframe` on a different origin.

[4] https://owasp.org/www-community/attacks/Clickjacking
[5] https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html

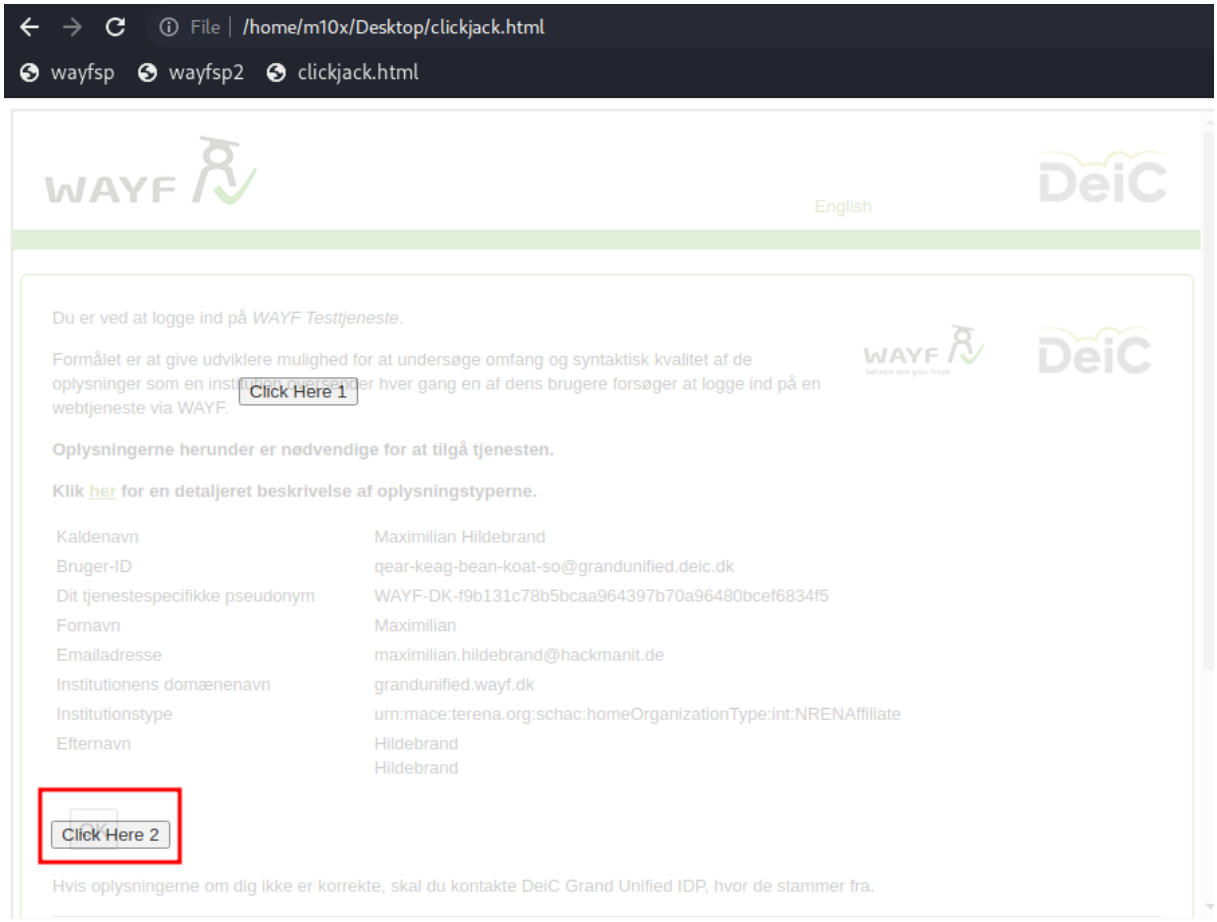

Figure 3: A victim could be tricked into unknowingly confirming the consent-page if it is embedded in a transparent `iframe`. In a real attack the `iframe` would be completely transparent.

6.3 M02 Insufficient Validation of "Audience" Information

Exploitability Metrics		**Impact Metrics**	
Attack Vector (AV)	**Network**	Confidentiality Impact (C)	**Low**
Attack Complexity (AC)	**High**	Integrity Impact (I)	**Low**
Privileges Required (PR)	**None**	Availability Impact (A)	**None**
User Interaction (UI)	**Required**	Scope (S)	**Changed**
Subscore: **1.6**		Subscore: **2.7**	

Overall CVSS Score for M02 : 4.7

General Description. Both ID tokens and SAML assertions usually contain information about the "audience" of the token / assertion: ID tokens contain an `aud` claim and SAML assertions contain an `<Audience>` element in `saml:Assertion/saml:Conditions/saml:AudienceRestriction`. This claim /

element names the intended recipient of the token / assertion.

The recipient of a token / an assertion should validate that the audience information names the recipient and reject tokens / assertions with audience information naming a different party. In addition, tokens / assertions without audience information should be rejected. The OIDC standard mandates that ID tokens contain an `aud` claim [1, Section 2], while the SAML schema defines the `<AudienceRestriction>` element as optional.[6]

Weakness. IdP_W supports both OIDC and SAML to delegate the authentication of the user to another IdP. After this IdP has authenticated the user, it sends an ID token or SAML assertion to IdP_W. On reception IdP_W verifies the signatures of these tokens / assertions and validates the information contained in them.

IdP_W correctly rejects tokens / assertions which contain audience information naming a party other than IdP_W as the intended recipient (e.g., `https://hackmanit.de`). However, IdP_W accepts tokens / assertions which do not contain valid audience information in the following cases:

- ID tokens without any `aud` claim. In Listing 10 an example ID token which was accepted by IdP_W is given.
- ID tokens with an `aud` claim set to `https://wayfsp.wayf.dk`. Even when IdP_W is technically running on the same machine as the SP it must be treated as a different entity and must not accept ID tokens which are intended for the SP.
- ID tokens with `aud` claims set to values of the following types: empty string, JSON array, JSON object, boolean, number
- SAML assertions without any `<AudienceRestriction>` element.
- SAML assertions with an `<Audience>` element containing an empty string.


```
[...].{
   "exp": 1682071763,
   "iat": 1682070863,
   "auth_time": 1682061463,
   "jti": "2f976bea-1929-4f68-a7c9-9c14c83d035f",
   "iss": "https://ec2-13-53-188-80.eu-north-1.compute.amazonaws.com/realms/IdP-OIDC",
   "sub": "e2565a8c-f7e5-40ba-887f-52138c14ee15",
   "typ": "ID",
   "azp": "https://wayf.wayf.dk",
   "nonce": "_gD6aFtwOChAk4BPfnOQKe8hfuL7d",
   "session_state": "1cfacb23-e5c9-4dff-b6df-21593fcd4299",
   "acr": "0",
   "sid": "1cfacb23-e5c9-4dff-b6df-21593fcd4299",
   "email_verified": false,
   "displayName": [
      "Karsten Meyer zu Selhausen"
   ],[...]
}.[...]
```


Listing 10: ID token accepted by IdP_W despite a missing `aud` claim (*condensed*).

[6] http://docs.oasis-open.org/security/saml/v2.0/saml-schema-assertion-2.0.xsd

Countermeasures. We recommend validating the audience information contained in ID tokens and SAML assertions strictly. On reception IdP_W should make sure that it is the intended recipient of the token / assertion by implementing the following measures:

1. Validate that ID tokens contain an `aud` claim.
2. Validate that the `aud` claim contains the client ID of IdP_W.
3. Validate that SAML assertions contain an `<AudienceRestriction>` and an `<Audience>` element.
4. Validate that the `<Audience>` element contains the identity of IdP_W.
5. Reject ID tokens with a different or missing `aud` claim.
6. Reject SAML assertions with different or missing audience information.

Retest. We can confirm that this weakness was successfully fixed. If the `<Audience>` element in a SAML assertion does not contain the expected value or no `<AudienceRestriction>` element exists at all, an error is raised: `HTTP/1.1 500 Internal Server Error [...] Audience mismatch "" != "https://wayf.wayf.dk"`

Further, if the `aud` claim of an ID token does not contain the expected value, a similar error is raised. If the `aud` claim is missing or set to a datatype other than string an error is raised, as well: `HTTP/1.1 502 Bad Gateway [...] The server returned an invalid or incomplete response.`

7 Recommendations

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

7.1 R01 Adding 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_W frontend does not use the security-related headers `X-Frame-Options`, `Content-Security-Policy`, `X-Content-Type-Options`, `X-XSS-Protection`, and `Strict-Transport-Security`.

The two headers `X-Frame-Options` and `Content-Security-Policy` are used to prevent attacks like clickjacking.[7] The combination of these two headers is important to protect users of both outdated and modern browsers. The headers can be used to specify whether a Web page is generally forbidden to be included as a frame (see Listing 11), whether it is allowed within the same origin (see Listing 12), or whether only certain hosts are allowed to do so (see Listing 13). Due to the fact that these two headers are not set, a clickjacking attack was possible (see M01).

Likewise, the content security policy (CSP) serves to mitigate the risk of content injection attacks. A missing or insecure configuration of the CSP can lead to an attacker being able to execute attacks such as XSS without restriction. A CSP can restrict this by, for example, disallowing inline scripts or insecure JavaScript functions, such as `eval()`. 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.

```
1 X-Frame-Options: deny
2 Content-Security-Policy: frame-ancestors 'none'
```

Listing 11: Strict clickjacking protection, which strictly prohibits the embedding of the Web page as a frame.

```
1 X-Frame-Options: sameorigin
2 Content-Security-Policy: frame-ancestors 'self'
```

Listing 12: Limited clickjacking protection, allowing the Web page to be included as a frame within the same origin.

[7] https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html

```
Content-Security-Policy: frame-ancestors 'self' *.somesite.com
```

Listing 13: Limited clickjacking protection that allows embedding the Web page as a frame from the mentioned hosts. `X-Frame-Options` unfortunately does not provide a satisfactory solution for this.

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.[8] 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.

The `X-Content-Type-Options: nosniff` header can be used by servers to forbid the browser from guessing a media type (MIME Type) of the resource. This behavior is known as "MIME sniffing" and allows the browser to guess the correct MIME Type by looking at the content of the resource. If the header is not present, MIME sniffing could allow the browser to transform non-executable content into executable content, thus enabling XSS attacks.[9]

The `Strict-Transport-Security` header (also called HSTS) requests the browser to access the Web page only using TLS or HTTPS in the future. This helps to prevent MitM attacks. An example configuration is shown in Listing 14.

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

Listing 14: Example configuration of HSTS.

Recommendation. We recommend adding the missing HTTP headers to the HTTP responses of the IdP_W frontend as described above:

- `Content-Security-Policy` with the directives:
  - `frame-ancestors 'none'`
  - `default-src` with an appropriate value for the application.[10]
- `X-Frame-Options: deny`
- `X-XSS-Protection: 0`
- `X-Content-Type-Options: nosniff`
- `Strict-Transport-Security: max-age=31536000; includeSubDomains`

[8] https://www.chromium.org/developers/design-documents/xss-auditor
[9] https://infosec.mozilla.org/guidelines/web_security#x-content-type-options
[10] For examples see: https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html#csp-sample-policies

Retest. We can confirm that this recommendation was fulfilled for the most part (see Listing 15). The three HTTP headers `X-XSS-Protection`, `X-Content-Type-Options`, and `Strict-Transport-Security` are now returned as recommended. The other two recommended HTTP headers `Content-Security-Policy` and `X-Frame-Options` are set to a the more lax recommendation, which prevents the IdP_W frontend from being included in a frame on a different origin. However, the `Content-Security-Policy` does not contain any source directive—contrary to the recommendation. The source directives, such as `default-src` or `script-src`, could serve as defense-in-depth mechanisms and mitigate possible XSS vulnerabilities.

```
1 Strict-Transport-Security: max-age=31536000; includeSubDomains
2 X-Content-Type-Options: nosniff
3 X-XSS-Protection: 0
4 Content-Security-Policy: frame-ancestors 'self'
5 X-Frame-Options: sameorigin
```

Listing 15: Security-related headers returned by IdP_W, after the recommendation was fulfilled for the most part.

7.2 R02 Improve the Replay Attack Protection

General Description. A SAML response contains several security parameters which mitigate replay attacks. The following security relevant parameters were not validated by IdP_W:

Response.ID Replay attacks are prevented by storing the unique ID of processed responses for a limited amount of time, for example, 30 minutes. Since the SAML response was not signed, this check could be skipped.

Assertion.ID Replay attacks are prevented by storing the unique ID of processed assertions for a limited amount of time, for example, 30 minutes.

Although these two security-relevant parameters are not validated, we were unable to perform a successful cross-site request forgery (CSRF) attack for the following reasons:

1. Step 2 of the flow (see: Figure 1) sets a cookie that must be present in step 6 and is validated by IdP_W. The cookie's name is random and the content is a HMAC hashed with a secret key.
2. The `InResponseTo` parameter is validated so that a SAML response cannot be redeemed for another SAML request.
3. The SAML response is only valid for a few minutes due to the `<Conditions>` element.

However, we recommend validating the two `ID` parameters as a defense-in-depth mechanism.

Recommendation. We recommend caching the `ID` of the SAML assertion and SAML response. The caching period can be a predefined value or it can be computed dynamically in dependence of the defined time limitations contained within the `<SubjectConfirmationData>`

or `<Conditions>` elements. Multiple redemptions of a SAML response or assertion are detected by comparing the stored `IDs` with the received ones.

Retest. This recommendation was not implemented at the time of the retest.

7.3 R03 Wrong Conversion of XML/JWT Encryption Algorithms

General Description. The "goxml" library implements a conversion from XML Encryption algorithms to JSON Web Algorithms (JWA). The encryption methods are stored in an encryption map:[11]

```
KeyEncryptionMethods = map[string]keyEncParams{
  "http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p": {"http://www.w3.org/2000/09/xmldsig#sha1", "RSA1_5"},
  "http://www.w3.org/2009/xmlenc11#rsa-oaep": {"http://www.w3.org/2001/04/xmlenc#sha256", "RSA-OAEP-256"},
  "http://www.wayf.dk/2009/xmlenc11#rsa-oaep-sha1": {"http://www.w3.org/2000/09/xmldsig#sha1", "RSA-OAEP"},
}
```

Listing 16: Key encryption map converting XML algorithms to JWT algorithms.

According to the implementation, an RSA-OAEP XML Encryption algorithm should be converted to an RSA PKCS#1 v1.5 algorithm. This is incorrect and should be fixed. See also: https://www.rfc-editor.org/rfc/rfc7518#appendix-A.2

Recommendation. We recommend the following modification, which makes the usage of the custom `wayf.dk` namespace superfluous:

```
KeyEncryptionMethods = map[string]keyEncParams{
  "http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p": {"http://www.w3.org/2000/09/xmldsig#sha1", "RSA-OAEP"},
  "http://www.w3.org/2009/xmlenc11#rsa-oaep": {"http://www.w3.org/2001/04/xmlenc#sha256", "RSA-OAEP-256"},
}
```

Listing 17: Corrected key encryption map for algorithm conversion.

Retest. We can confirm that this recommendation was successfully fulfilled. The `KeyEncryptionMethods` map was modified as recommended.[12]

[11] https://github.com/wayf-dk/goxml/blob/3357d6e073b2a2e6e539ac57b879267e9453bfb9/crypt.go#L46
[12] https://github.com/wayf-dk/goxml/commit/2a692069d42eadae3c5287db9dd969480fc6698f

7.4 R04 Insufficient Validation of `<Signature>` Elements' Count

General Description. The signature verification logic implemented in the "goxml" library first attempts to ensure that the XML message only contains one XML Signature element. This is done with the following code:[13]

```
  signaturelist := xp.Query(context, "ds:Signature[1]")
if len(signaturelist) != 1 {
  return fmt.Errorf("no signature found")
}
signature := signaturelist[0]
```

Listing 18: Part of the signature verification logic to ensure only one XML Signature element is present.

However, the XPath expression in the first line of the code always selects only the first `<Signature>` element, which makes the check for the correct number of elements superfluous; the query string will always return a list containing one `<Signature>` element.

Recommendation. Given the later strict XML Signature handling, we do not think that this small issue will affect the security of the XML Signature. Nevertheless, we recommend fixing this issue.

Retest. We can confirm that this recommendation was successfully fulfilled.[14]

[13] https://github.com/wayf-dk/goxml/blob/3357d6e073b2a2e6e539ac57b879267e9453bfb9/crypt.go#L103
[14] https://github.com/wayf-dk/goxml/commit/4c360423f3e0c9a36f9cfe663f4a71cb2257a90a

8 Information

In the following sections, we describe observations of unusual configurations and possibly unwanted behavior of the tested system.

8.1 I01 Automatic Modification of the `eduPersonPrincipalName` Claim

General Description. In the scenario of the penetration test each IdP is restricted to a "scope" (e.g., `@hackmanit.de`) and not allowed to issue ID tokens or SAML assertions for different scopes. The scope is the suffix of the `eduPersonPrincipalName` field which is used to identify the user.

IdP_W rejected SAML assertions issued by IdP_H if they contained an `eduPersonPrincipalName` field for a scope other than `@hackmanit.de`.

However, when IdP_H issued ID tokens with an `eduPersonPrincipalName` claim for a scope other than `@hackmanit.de` IdP_W accepted the ID token. An example of an ID token with the `eduPersonPrincipalName` set to `toyq-wiid-soeh-paal-ho@grandunified.deic.dk` is depicted in Listing 19. IdP_W issued a SAML assertion to the SP afterwards. For this new assertion IdP_W takes the value of the `eduPersonPrincipalName` claim and replaces the scope of this value with the scope IdP_H is restricted to. An example of a SAML assertion with the `eduPersonPrincipalName` set to `toyq-wiid-soeh-paal-ho@hackmanit.de` issued by IdP_W is given in Listing 20.

This behavior is not considered as a weakness, because the `eduPersonPrincipalName` value that the SP receives contains the scope IdP_H is restricted to. However, this behavior for ID tokens does not seem to be intended and differs from the handling of SAML assertion. The cause of this behavior might be a workaround introduced during the penetration test specifically for IdP_H.

```
{
   "exp": 1682085197,
   "iat": 1682084297,
   "auth_time": 1682082482,
   "iss": "https://ec2-13-53-188-80.eu-north-1.compute.amazonaws.com/realms/IdP-OIDC",
   "aud": "https://wayf.wayf.dk",
   "sub": "e2565a8c-f7e5-40ba-887f-52138c14ee15",
   "nonce": "_RkHyAHqS2QZcAeIrnP8WSX14aMJK",
   "sn": [
      "Meyer zu Selhausen"
   ],
   "eduPersonPrincipalName": [
      "toyq-wiid-soeh-paal-ho@grandunified.deic.dk"
   ],[...]
}
```

Listing 19: Example of an ID token issued by IdP_H with an `eduPersonPrincipalName` claim for a not allowed scope *(condensed)*.

```
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" Version="2.0" ID="
    _OX0U_aOMh2G9ixZeoQvG_LB9ISuO" IssueInstant="2023-04-21T13:38:38Z">
<saml:Issuer>https://wayf.wayf.dk</saml:Issuer>
<saml:AttributeStatement>
   <saml:Attribute Name="cn" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:
       basic" FriendlyName="cn">
      <saml:AttributeValue>Karsten Meyer zu Selhausen</saml:AttributeValue>
   </saml:Attribute>
   <saml:Attribute Name="eduPersonPrincipalName" NameFormat="urn:oasis:names:tc:SAML
       :2.0:attrname-format:basic" FriendlyName="eduPersonPrincipalName">
      <saml:AttributeValue>toyq-wiid-soeh-paal-ho@hackmanit.de</saml:AttributeValue>
   </saml:Attribute>
   <saml:Attribute Name="sn" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:
       basic" FriendlyName="sn">
      <saml:AttributeValue>Meyer zu Selhausen</saml:AttributeValue>
   </saml:Attribute>
   [...]
</saml:AttributeStatement>
</saml:Assertion>
```

Listing 20: Example of a SAML assertion with a modified `eduPersonPrincipalName` value issued by IdP_W *(condensed)*.

Recommendation. We recommend investigating the behavior described above and determine whether it is intended. We recommend handling ID tokens in a similar way as SAML assertions. This means IdP_W should reject ID tokens with an `eduPersonPrincipalName` claim for a scope other than the one the issuer of the ID token is restricted to.

Retest. We can confirm that issued ID tokens with an `eduPersonPrincipalName` claim for a not allowed scope are now rejected. This behavior is consistent with SAML assertions which contain an `eduPersonPrincipalName` claim for a not allowed scope.

8.2 I02 Improvable Handling of Unsupported `response_type` Values

General Description. When IdP_W is invoked as an IdP in an OIDC protocol flow it only supports the value `id_token` for the `response_type` parameter. However, other values for the parameter are not rejected. The following values were tested: `token+id_token`, `id_token+token`, `token`, `code`, `code+id_token`, `id_token+code`, `code+token`, `code+token+id_token`, `code+id_token+token`

In all cases IdP_W accepted the request and forwarded the user to IdP_G or IdP_H. Afterwards, IdP_W issued a SAML response instead of an OIDC response. For the test SP used during the penetration test this resulted in a successful login because the SP supports both SAML and OIDC. In a real-life scenario an OIDC relying party usually does not support SAML and even when it does it should reject SAML responses received when it started an OIDC protocol flow.

Recommendation. We recommend altering the implementation of IdP_W to reject unsupported values for the `response_type` parameter and only returning SAML responses when answering SAML requests.

Retest. We can confirm that the recommendation was successfully fulfilled. Values other than `id_token` for the `response_type` parameter are now rejected.[15] [16]

[15] https://github.com/wayf-dk/gosaml/commit/12537f729bd4053a31378dca79b2da8cce35958f
[16] https://github.com/wayf-dk/gosaml/commit/fcc5d19418a4eed228f28fc3792da42a974938b9

9 Further Evaluations

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

9.1 SAML

SAML Request Specific Tests. The following elements and attributes in the SAML request to IdP_W were manipulated:

- `Destination`:
  The `Destination` attribute is neither validated nor processed. Thus, it can be set to an arbitrary value or be removed, without affecting the flow.
- `AssertionConsumerServiceURL`:
  If the `AssertionConsumerServiceURL` attribute is not present the value from the metadata stored for the SP is used. If a value is present it must match the value from the stored metadata. Otherwise an error message is returned: `["cause: AsssertionConsumerService, AsssertionConsumerServiceIndex, ProtocolBinding combination not found in metadata", "acs: https://wayfsp.wayf.dk/ACSFOOBAR", "acsindex:", "binding:urn :oasis:names:tc:SAML:2.0:bindings:HTTP-POST"]`.
- `Issuer`:
  The `Issuer` element must be specified, otherwise an error is returned: `no issuer found in SAMLRequest/SAMLResponse`. If the value is replaced with another valid issuer an error message is returned as well: `["cause: AsssertionConsumerService, AsssertionConsumerServiceIndex, ProtocolBinding combination not found in metadata","acs: https://wayfsp.wayf.dk/ACSFOOBAR","acsindex:","binding:urn :oasis:names:tc:SAML:2.0:bindings:HTTP-POST"]`. This behavior is expected, because the `Issuer` and the `AssertionConsumerServiceURL` do not match. If an unknown issuer is used, the following error is returned: `["cause :Metadata not found", "err:Metadata not found", "key:https ://75acm5ug34jbgi6sowefh5uv7mdd18px.oastify.com", "table:sp "]`.
- `ProtocolBinding`:
  The `ProtocolBinding` attribute was set to different valid bindings, as well as, arbitrary strings (e.g., `HACKMANIT`). IdP_W seems to ignore this attribute and always uses the POST binding for the SAML response to the SP.

It was also checked if `Destination`, `AssertionConsumerServiceURL`, or `Issuer` were vulnerable to server-side request forgery (SSRF) attacks. URLs placed in these fields were not invoked by IdP_W. Thus, all three elements behaved as expected and provided no potential for attack.

SAML Response Specific Tests. The following elements and attributes in the SAML response to IdP_W were tampered with, as they were not protected by a signature.

- `Issuer`: Similar to the SAML request, an error message is returned if the `Issuer` does not reference the IdP which issued the SAML response.
- `Status`: If the value is changed, an error message is returned: `["cause: check failed", "check: count(/samlp:Response/saml:Assertion)= 1"]`.

Using our own IdP_H, we were able to generate validly signed SAML responses with arbitrary contents. The following tests were conducted using IdP_H:

- `InResponseTo`: The `InResponseTo` attribute is used to protect against CSRF attacks. It occurs once in the SAML response and once in the assertion. IdP_W validates both attributes correctly and returns an error if they are different, one is missing, or both are manipulated in the same way.
- `eduPersonPrincipalName`: IdP_W validates that the `eduPersonPrincipalName` user attribute is within the allowed scope of the IdP that issued the SAML response. If this value is outside the scope of the IdP, an exception is thrown. It was not possible to bypass this validation, e.g. by using different encodings or by manipulating other elements such as `StatusCode`.
- `Issuer`: The `Issuer` element occurs once in the SAML response and once in the assertion. IdP_W validates both elements correctly and throws an error if they are different or one is missing. The element was also tested with invalid values and empty strings and the element was removed. None of the attack vectors could be exploited successfully.
- `Audience`: The element was tested with invalid values and empty strings. It was also tested to delete the element. Except for an empty or removed `Audience` element (see M02) none of the attack vectors could be exploited successfully. See also "Recipient / Audience Information Validation" below.
- `StatusCode`: The element was tested with invalid values and empty strings. It was also tested to delete the element. None of the attack vectors could be exploited successfully.
- Timestamps: Multiple timestamps in the SAML assertion were manipulated in various ways. They were set to future and past times, to see if assertions are accepted although they are expired. Timestamps were set to invalid values and deleted as well. None of the attack vectors could be exploited successfully. The following timestamps were tested:
  - `IssueInstant` attribute in `Response` element
  - `IssueInstant` attribute in `Assertion` element
  - `NotOnOrAfter` attribute in `SubjectConfirmationData` element
  - `NotBefore` attribute in `Conditions` element
  - `NotOnOrAfter` attribute in `Conditions` element
  - `AuthnInstant` attribute in `AuthnStatement` element

Recipient / Audience Information Validation. SAML responses can contain information about the intended recipient in multiple elements/attributes. The `Audience` element is not sufficiently validated (see M02). The `Destination` attribute of the `Response` element is compared to the `Recipient` attribute inside the `SubjectConfirmationData` element. IdP$_W$ rejects messages if these attributes do not match or if one of these attributes is not present.

Replay Attacks. SAML assertions can be sent to IdP$_W$ multiple times in the same browser session, as long as they are not expired. This is possible because the `ID` attributes of the `Response` and `Assertion` elements are not validated by IdP$_W$ (see R02). However, more relevant replay attacks which try to use a SAML assertion in a different browser session could not be executed successfully. IdP$_W$ validates the `InResponseTo` attributes of the `Response` and `Assertion` elements correctly. The two attributes must be equal and match the ID bound to the user's session cookie (`SSO2-...`).

General SAML Tests. The following general tests for SAML messages were executed:

- Multiple XML elements with the same name:
  If multiple XML elements have the same name, only the last occurrence is used. If this behavior would not have been consistent, and, for example, one element would have been used for validation and the other for further processing, an attack might have been possible.
- XML Entities: IdP$_W$ decodes default XML entities in SAML responses and processes them correctly depending on the context. When a SAML response is sent, the content is also XML encoded. When an ID token is sent, the content taken from the SAML response is escaped if it contains JSON special characters.
- Comments:
  If comments are not handled properly in terms of the digest calculation this can be exploited by an attacker. IdP$_W$ removes comments when processing XML messages and correctly does not include them in the digest calculation (see also Node Splitting)

Node Splitting. IdP$_W$ supports the canonicalization method which is a prerequisite for node splitting attacks. Nevertheless, it was not possible to successfully execute node splitting attacks against IdP$_W$. For example, a comment (`<!- ->`) was injected into the value of the `eduPersonPrincipalName` attribute in the SAML response to IdP$_W$. The comment did not result in node splitting and the comment was removed by IdP$_W$ before issuing a SAML response to the SP. Also Document Type Definitions (DTDs) could not be used for node splitting attacks. IdP$_W$ rejected messages which contain self-defined entities with an error message: `["cause:schema validation failed"]`

Document Type Definition (DTD) Based Attacks. The following DTDs based attacks were tested.[17] None of them was exploitable.

- Various Variants of XML External Entity (XXE) Attacks: IdP$_W$ does not support or process external entities or self-defines entities in general. HTTP requests were tested with all valid XML content types.

[17] The tests were based on this cheat sheet: https://web-in-security.blogspot.com/2016/03/xxe-cheat-sheet.html

- Recursive General Entities: Sending the payload resulted in an error stating that the schema validation failed.
- SSRF: IdP$_W$ did not send requests to the test URL.
- Different Techniques For Bypassing Restrictions of XXE: Sending the payload resulted in an error stating that the schema validation failed.
- XSLT Attacks: Sending the payload resulted in an error stating that the schema validation failed.
- XInclude Attacks: Sending the payload resulted in an error stating that the schema validation failed.

eXtensible Stylesheet Language Transformation (XSLT). It was not possible to make IdP$_W$ process XSLT payloads, for example to conduct SSRF attacks. One example payload is given in Listing 21. XSLT payloads were sent to IdP in different SAML/XML messages (e.g., directly in the message body or placed as a transformation inside of the signature element). IdP$_W$ did not invoke the URL contained in the payloads and rejected the messages with the error message: `["cause:schema validation failed"]`

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="doc">
<xsl:variable name="attackerUrl" select="'http://pqjb764ad2x6e0slkmkuqpopcgi763us.
    oastify.com/'"/>
<xsl:value-of select="unparsed-text($attackerUrl)"/>
</xsl:template>
</xsl:stylesheet>
```

Listing 21: Example payload for a SSRF attack using XSLT.

XML Signature Verification. SAML assertions sent to IdP$_W$ are usually signed using XML signature. IdP$_W$ verifies the signature and it was not possible to bypass this verification.

SAML assertions without a signature are rejected with the error message:[18] `["cause:encryption error"]`.

SAML assertions with invalid signatures or signatures signed by keys IdP$_W$ does not trust are rejected with the error message: `["cause:crypto/rsa: verification error","err:unable to validate signature"]`.

We injected multiple assertions into the message to test whether the server accepts any message with more than one `Assertion` element. The server always responded with `["cause: check failed","check: count(/samlp:Response/saml:Assertion)= 1"]` (even if one assertion was encrypted).

We injected multiple encrypted assertions. The server responded with: `["cause: encryption error"]`.

XML signature wrapping (XSW). After XSW was tested during the penetration tests conducted in 2018, XSW vectors were tested again semi-automatically. None of the tested attack

[18] Please note: While verifying a signature is cryptographically similar to encrypting a message this error message might be confusing and could be replaced with a more fitting one (e.g., `["cause:invalid or missing signature"]`).

vectors could be executed successfully and most of them were rejected with the error message: `["cause:ID mismatch","err:unable to validate signature"]`.

Dupe Key Confusion. Dupe key confusion is an XML signature bypass technique where multiple key identifiers are sent in the `KeyInfo` section. A system is vulnerable if it uses one key to verify the XML signature and the other to verify trust in the signing party. IdP$_W$ uses the first specified key to verify both the XML signature and the trust in the signing party, and is therefore not vulnerable to dupe key confusion. Our code review showed that the vulnerability was not present because all public keys and certificates are stored in the local metadata and the server ignores cryptographic material provided in the message.

Cryptographic Algorithms. We performed the following tests to evaluate the correctness of cryptographic algorithm processing, with the following results:

1. The usage of signing and digest algorithms is enforced using an explicit function. The algorithms are provided in the config.[19]
2. Only secure algorithms are used: RSA-OAEP and AES-GCM.
3. It was not possible to change algorithms or introduce any ciphertext modifications since the ciphertexts are authenticated. All attempts resulted in an encryption error. This applies for XML Encryption, as well as, for JWE.

Message Confusion. It was tested whether IdP$_W$ could be "confused" to accept genuine SAML messages as a replacement for different messages in the protocol flow. For example a SAML response issued by IdP$_W$ itself (#6 in Figure 1) was used in another protocol flow as SAML response to IdP$_W$ (#5 in Figure 1). IdP$_W$ rejected the response with an error message because itself is not present in its own list of trusted IdPs.

Another example was replacing the SAML response from IdP$_G$ with a valid SAML response issued by another IdP. IdP$_W$ rejected the response with an error message because the user's session cookie (`SSO2-...`) did not match the IDs in the SAML response: `http: named cookie not present`.

ID Spoofing. SAML assertions issued by IdP$_H$ containing an identity of a user of IdP$_G$ (e.g., `toyq-wiid-soeh-paal-ho@grandunified.deic.dk`) were rejected by IdP$_W$ with the error message: `["cause:security domain 'grandunified.deic.dk' does not match any scopes"]`.

Additionally manipulating the `Issuer` element of the assertion resulted in the error message: `["cause: check failed","check: /samlp:Response/saml:Issuer = /samlp:Response/saml:Assertion/saml:Issuer"]`.

Additionally manipulating the `Issuer` element of the response resulted in the error message: `["cause:crypto/rsa: verification error","err:unable to validate signature"]`.

This means IdP$_W$ was not vulnerable to such "simple" ID spoofing attacks. However, only one step more at the beginning of the attack allows an attacker to execute successful ID spoofing

[19] https://github.com/wayf-dk/gosaml/blob/master/gosaml.go#L2140

attacks (see C01). The attacker simply needs to select an honest IdP (e.g., IdP_G) at the beginning of the authentication flow and can use their own IdP_H to issue assertions about identities in the scope of the other IdP. IdP_W accepts the assertion because IdP_H is generally a trusted IdP and the identity in the assertion is in the expected scope.

9.2 OpenID Connect

IdP_W as OIDC IdP

The following parameters were tested when IdP_W is invoked with an OIDC authentication request.

`response_type`. IdP_W does not support values for the `response_type` parameter other than `id_token`. It was not possible to use different OIDC protocol flows with IdP_W. However, the handling of unsupported values for the `response_type` parameter could be improved (see I02).

`redirect_uri`. IdP_W seems to ignore this parameter. The parameter can be set to arbitrary values (incl. an empty string) or removed from the request. IdP_W always redirects the user to `https://wayfsp.wayf.dk/ACS` along with the issued ID token.

IdP_W as OIDC Relying Party

The following tests were conducted when OIDC is used by IdP_W to delegate the authentication of the user to IdP_H. IdP_W supports the OIDC implicit flow for this purpose.

Consent-Page. IdP_W shows a consent-page to allow the user to decide whether their data should be shared with the SP. This consent-page displays user data such as the user's name. When user data is added to the HTML source code of the consent-page it is escaped correctly. For example a user's name `Meyer zu Selhausen<script>alert(1)</script>` is converted to `Meyer <wbr>zu <wbr>Selhausen&lt;<wbr>script&gt;<wbr>alert(<wbr>1)<!--<wbr-->script&gt;<wbr>` before it is added to the source code.

Similarly, user data is escaped when it is added to JSON data stored in a JavaScript variable on the consent-page (e.g., `"` is converted to `\"`).

`state` Parameter. The `state` parameter is sent by IdP_W to IdP_H in the authentication request. IdP_H returns the same parameter in the authentication response to IdP_W.

IdP_W uses an empty string as the value when the OIDC implicit flow is used. This parameter is superfluous in this case. In the implicit flow the `nonce` parameter already provides the same protection as the `state` parameter. Therefore, the `state` parameter could be removed completely.

ID Token Signature Verification. IdP_W validates ID tokens received from IdP_H. The signature of the ID tokens is verified correctly. ID tokens with invalid signatures, without any signature, or with a valid signature created with an untrusted key are rejected with the error message

`["cause:jwtVerify failed"]`. IdP$_W$ was not vulnerable to SSRF attacks; it did not invoke URLs placed in the header of ID tokens. Unsigned ID tokens which use the `none` algorithm are rejected with the error message `["cause:Unsupported alg: none"]`.

ID Token Claim Validation. The validation of the ID token has been examined for the following claims:

- `iss`: The value of the claim is validated and cannot be set to an arbitrary string. The claim cannot be removed either. It must reference an IdP registered at IdP$_W$.
- `aud`: The value of the claim is validated and cannot be set to an arbitrary URL or string. However, the validation is insufficient (see M02).
- `nonce`: The `nonce` is correctly validated. ID tokens without a `nonce` claim or with an arbitrary string value are rejected. Also valid values from other browser session's are rejected because the `nonce` value is bound to the session cookie (`SSO2-...`). After an ID token is received by IdP$_W$ the cookie is removed from the user's browser. This prevents replay and possibly CSRF attacks.
- `iat` / `exp`: The timestamps are validated by IdP$_W$. Removing the claims results in the implementation treating their values as `0` which causes the validation to fail. The timestamp in `iat` must not be in the future and the timestamp in `exp` must not be in the past. IdP$_W$ accepts timestamps with a small margin for error, likely to deal with clock skew.

Mapping Between ID Tokens and SAML Assertions. IdP$_W$ supports the use of OIDC and SAML both when it is invoked by a SP / client or when it delegates the authentication of the user to another IdP.

This means IdP$_W$ needs to map information from a SAML assertion issued by another IdP to an ID token and the other way around. It was not possible to identify any weaknesses in this mapping process – only a probably undesired behavior (see I01). When a claim in an ID token contains characters with a special purpose in an XML setting, IdP$_W$ escaped these characters correctly before adding them to a SAML assertion. Similarly, IdP$_W$ escaped JSON special characters contained in SAML assertions before adding them to ID tokens. It was not possible to influence user attributes such as `schacHomeOrganization`, `eduPersonTargetedID`, or `schacHomeOrganizationType` by adding them to a SAML assertion or ID token sent to IdP$_W$.

9.3 Miscellaneous

Headers. Various headers such as `X-Forwarded-For` and `Origin` were specified, which can cause a change in behavior and thus provide an attack surface. However, IdP$_W$ did not show any vulnerable behavior.

Cookies. It is always recommended to set the security based cookie flags `HttpOnly`, `Secure`, and if possible `SameSite=Strict`. IdP$_W$ sets the two cookies `SSO2-...` and `SLO`. Both cookies have the `HttpOnly` and `Secure` flags set as recommended. However, `SameSite=None` is also set, which provides no protection against CSRF attacks. This is common for SAML flows, as the `SameSite` flag, when set to `Lax` or `Strict`, disallows cookies to be sent with

POST requests. This prevents the SAML flow from working correctly. In this case, `SameSite=None` is not dangerous because CSRF protection is provided at the SAML level, e.g. by the `InResponseTo` element.

Code Review. During the penetration test, the relevant code of the GitHub repositories `wayf-dk/wayfhybrid`[20], `wayf-dk/gosaml`[21] and `wayf-dk/goxml`[22] was checked. Based on the source code, no vulnerabilities or recommendations were found that were not previously discovered by dynamic testing.

[20] https://github.com/wayf-dk/wayfhybrid/tree/75a992b739ca1b0bc4a272567ff2621e020d29bd
[21] https://github.com/wayf-dk/gosaml/tree/1b6ffb41f12f0a4e0d638b2ed22fa1c238a56a81
[22] https://github.com/wayf-dk/goxml/tree/283f4786a4cea26097fbcc3a7e4a0751c40f8512

10 References

[1] N. Sakimura et al. *Openid Connect Core 1.0*. OpenID Foundation, Nov. 2014. url: http://openid.net/specs/openid-connect-core-1_0.html.