Protokollübergreifendes SSO

SAML und OIDC

71. DFN-Betriebstagung

24. September 2019

David Hübner
DAASI International GmbH

Agenda

- Vergleich SAML & OIDC
- Lösung 1: Gemeinsames Login-Modul
  Gluu Server
- Lösung 2: Modularer Proxy
  Satosa / didmos Authenticator
- Lösung 3: Protokollübergreifendes Session-Handling
  OIDC-Erweiterung für Shibboleth IDP
- Fazit, Vor- & Nachteile

DAASI International

Vergleich SAML & OIDC

DAASI International

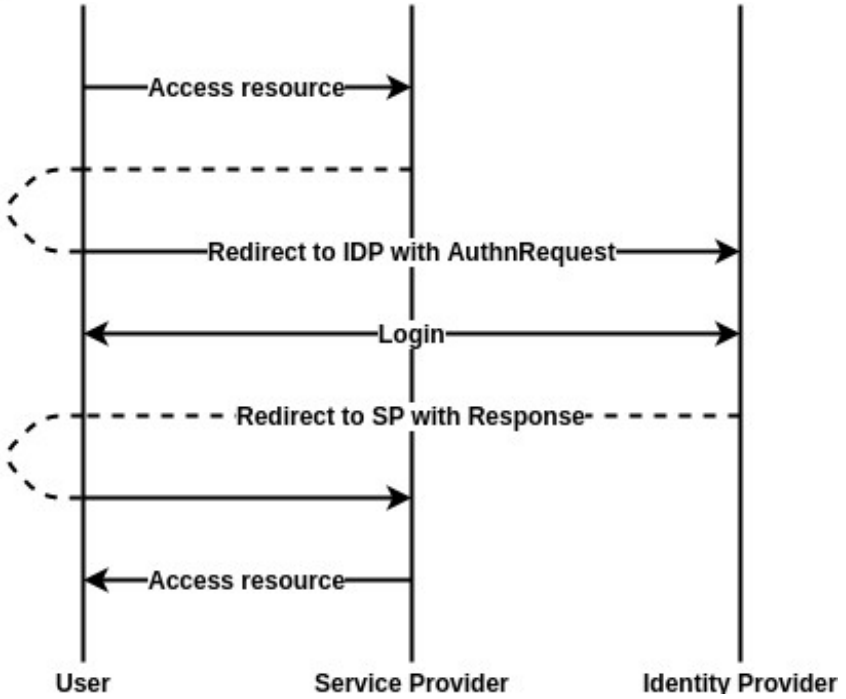
Vergleich SAML & OIDC

- SAML 2.0 existiert seit 2005 und ist eine etablierte Lösung
- Shibboleth (IDP & SP) sind vor allem im R&E-Bereich weitverbreitete Implementierungen des SAML-Protokolls
- Es wird primär das Web Browser SSO Profile eingesetzt

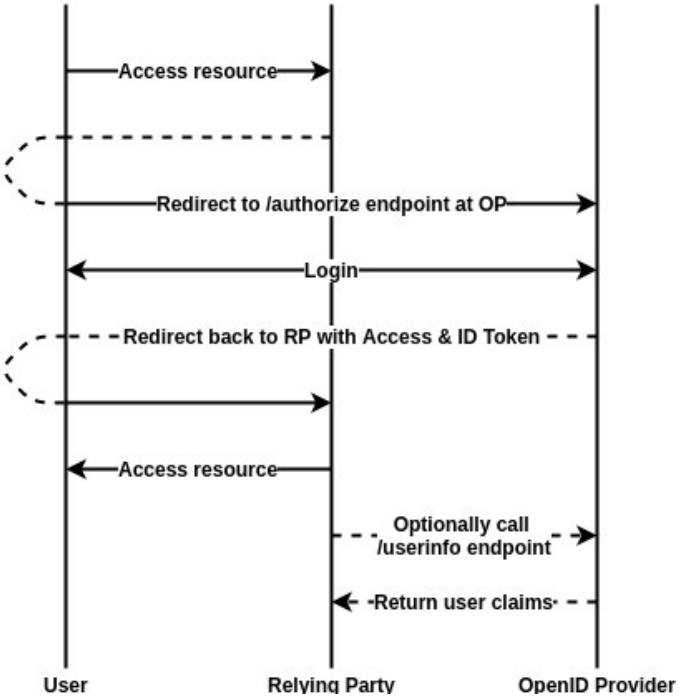
- OIDC setzt als modernere und schlankere Alternative auf dem OAuth2-Protokoll auf
- Durch die Verbreitung v.a. bei mobilen Endgeräten (teilweise auch OAuth2) und die Unterstützung von großen Herstellern v.a. im Cloud-Bereich gewinnt OIDC in den vergangenen Jahren an Verbreitung
- Es werden primär der Implicit & Authorization Code Flow eingesetzt

DAASI International

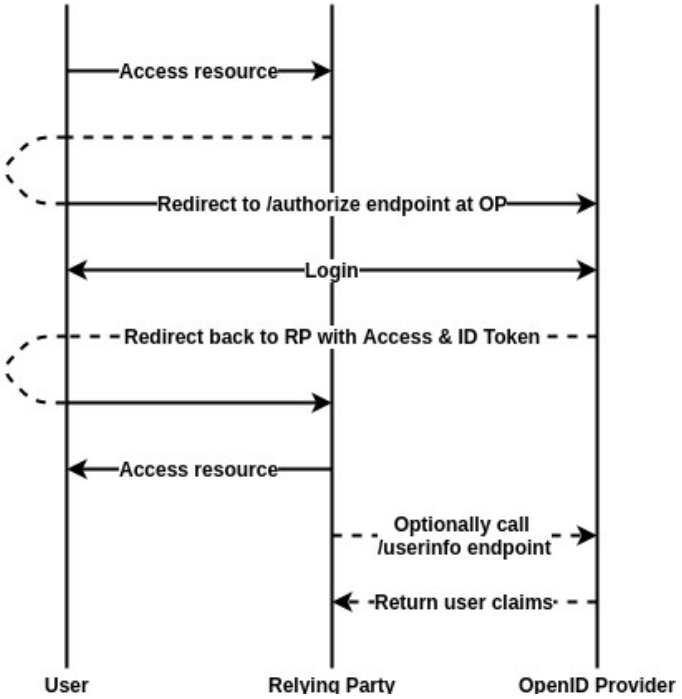
SAML Web Browser SSO



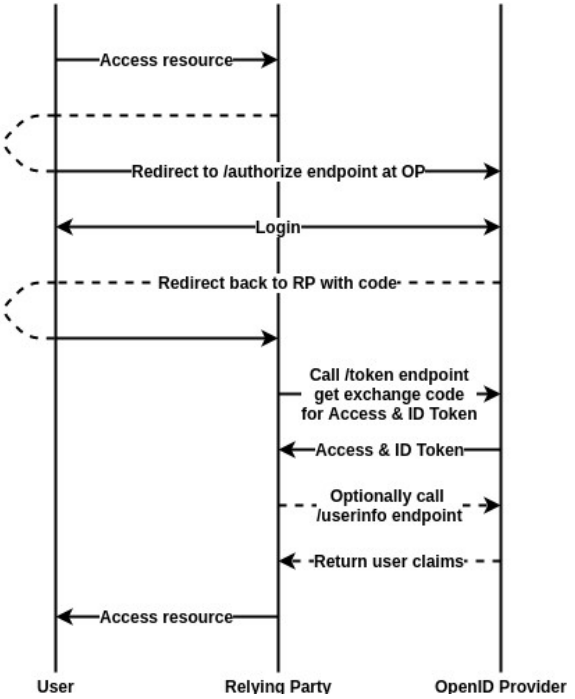
OIDC Implicit Flow



OIDC Implicit Flow



OIDC Authorization Code Flow



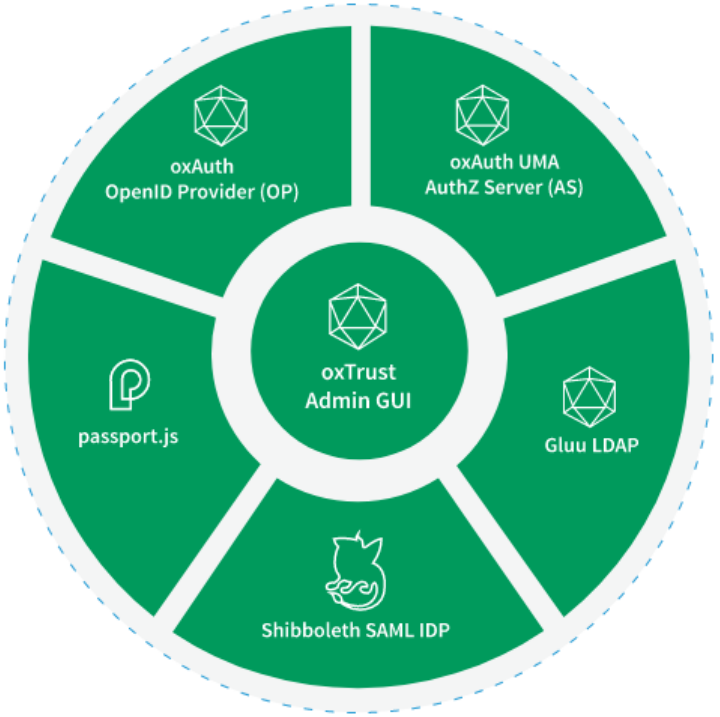
Terminologie

- Terminologie:
  - IDP = OP (OpenID Provider)
  - SP = RP (Relying Party)
- In OIDC werden verschiedene Flows definiert:
  - Implicit Flow: Access Token und ID Token werden direkt an die RP geschickt (was nicht unbedingt sicher ist, aber das ist ein anderes Thema)
  - Authorizatzion Code Flow: Tokens müssen von der RP extra abgefragt werden
- In OIDC werden Endpunkte definiert:
  - /authorize – Wird mit einem Aufruf analog zum AuthnRequest angesprochen
  - /token – Im Authorization Code Flow tauscht die RP hier den code gegen die Tokens
  - /userinfo – Zur Abfrage von weiteren User-Attributen (Claims)

Szenario 1: Nutzung eines gemeinsamen Login-Moduls

Beispiel: Gluu Server

Gluu Server - Übersicht



Gluu Server – Shibboleth IDP

- Gluu Server bindet einen vollständigen Shibboleth IDP ein und erlaubt die Konfiguration von „Standard-Einstellungen“ per oxTrust.
- Shibboleth nutzt oxAuth zur Authentifizierung und ermöglicht so SSO.

Gluu Server – oxTrust

- OxTrust ist das Admin-Interface von Gluu, welches eine umfangreiche Konfiguration per GUI ermöglicht.
  - Anbinden von OIDC/SAML Diensten
  - Systemkonfiguration
  - Verwaltung von Nutzern in Gluu („IDM light“)
  - Verwaltung von Attributen
  - Interceptor Scripts

Gluu Server – oxAuth

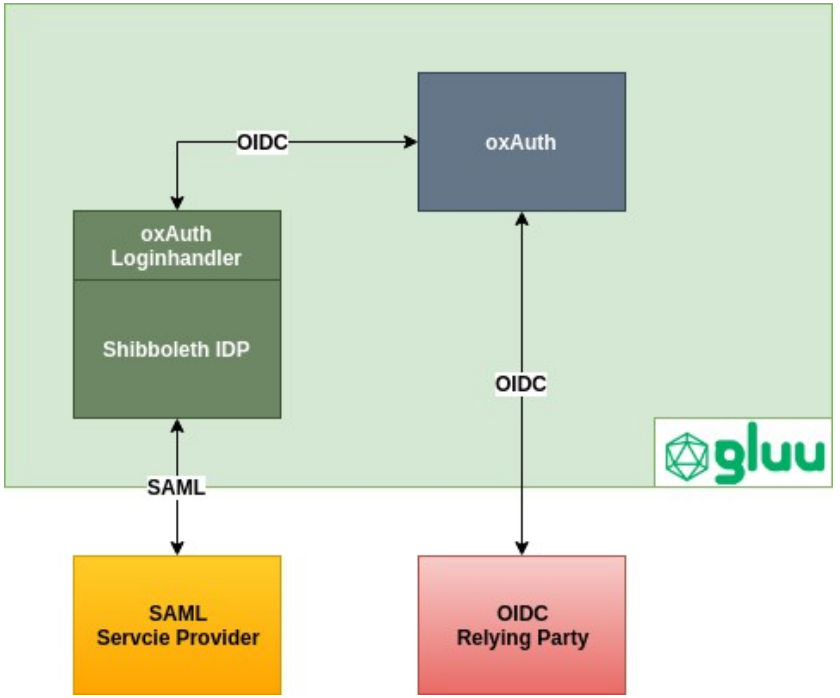
- oxAuth ist der OpenID Provider in Gluu Server und ermöglicht das Anbinden von Clients über OIDC.
- Weiterhin fungiert oxAuth auch als OAuth2.0 Authorization Server und kann Access Tokens für den Zugriff auf beliebige Resourcen ausstellen (OAuth 2.0 UMA)
- Die Clients, Scopes und Claims lassen sich über oxTrust konfigurieren.

Gluu Server – LDAP

- Gluu LDAP ist die Datenbank von Gluu und speichert Nutzer, Konfiguration, Sessions, etc.
- Nutzer können entweder direkt in Gluu angelegt werden oder über LDAP/AD Synchronisation bzw. SCIM API nach Gluu provisioniert werden.
- In zukünftigen Versionen wird als Alternative zu LDAP auch Couchbase als Storage-backend angeboten.

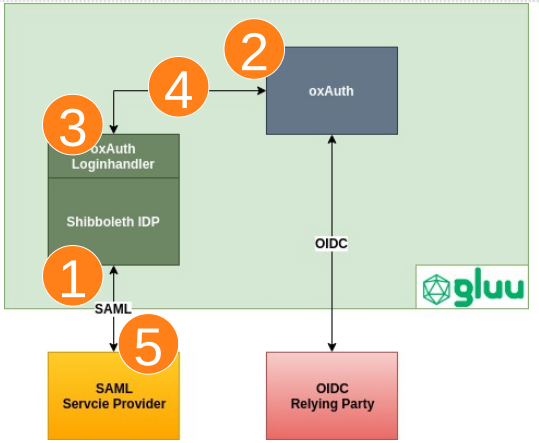
Authentifizierung in Gluu

- Die Authentifizierung findet in allen Protokollen an oxAuth (per OIDC) statt
- Dafür bindet Gluu in Shibboleth IDP einen Custom-Loginhandler ein, der sich per OIDC an oxAuth authentifiziert



Beispiel: Login Flow über Shibboleth IDP

1	GET	https://[redacted].org/idp/profile/SAML2/Redirect/SSO?SAMLRequest=fZJNb4IwGMe%2FCuldCgWcNkLC9DATt	SAML
	GET	https://[redacted].org/idp/profile/SAML2/Redirect/SSO;jsessionid=node07y814khv33sb1htijjkepbebx27.node0?execution=e1	
	GET	https://[redacted].org/idp/Authn/oxAuth?conversation=e1s1	
2	GET	https://[redacted].org/oxauth/restv1/authorize?response_type=code&client_id=%40%215BB4.1A50.B87F.C9F4%210001%2	
	GET	https://[redacted].org/oxauth/authorize.htm?scope=openid+email+user_name&response_type=code&redirect_uri=https%3A	
	GET	https://[redacted].org/oxauth/login.htm	
	POST	https://[redacted].org/oxauth/login.htm	
	GET	https://[redacted].org/oxauth/authorize.htm?scope=openid+email+user_name&response_type=code&redirect_uri=https%3A	
	GET	https://[redacted].org/oxauth/restv1/authorize?scope=openid+email+user_name&response_type=code&redirect_uri=https%3	
3	GET	https://[redacted].org/idp/Authn/oxAuth?code=8c6283a6-5375-4b42-b7ad-c61ad1833b4e&scope=openid+user_name+emai	
	GET	https://[redacted].org/idp/profile/SAML2/Redirect/SSO?execution=e1s1&_eventId_proceed=1	
5	POST	https://[redacted]:8443/Shibboleth.sso/SAML2/POST	SAML



Szenario 2: Nutzung eines modularen Proxys

Beispiel: Satosa / didmos Authenticator

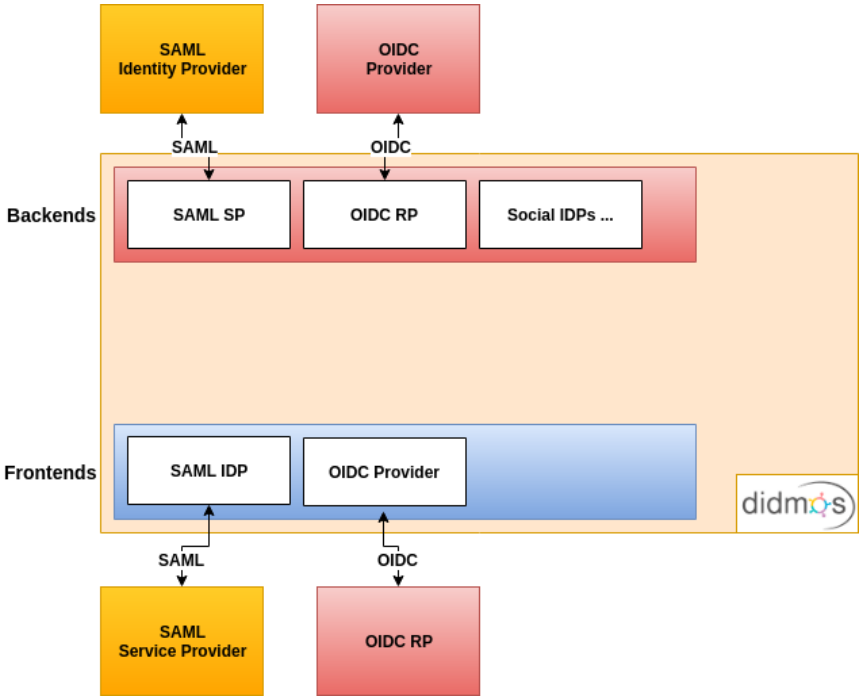
Einführung in Satosa und SSO-Proxies

- Transparente Übersetzung zwischen verschiedenen Protokollen (Usecase 1)
- Satosa ist ein modularer SSO-Proxy in Python
- Basiert auf verschiedenen Bibliotheken von IdentityPython
- Viele populäre Social/Corporate IDPs basieren auf OIDC oder OAuth2
- Damit lassen sich diese leicht in Satosa integrieren und mit einem vorhandenen SAML IDP kombinieren (Usecase 2)
- Abstraktion von Föderationsmetadaten für SPs, die nur an einen IDP angebunden werden können (Usecase 3)

DAASI International

Einführung in den Aufbau von Satosa

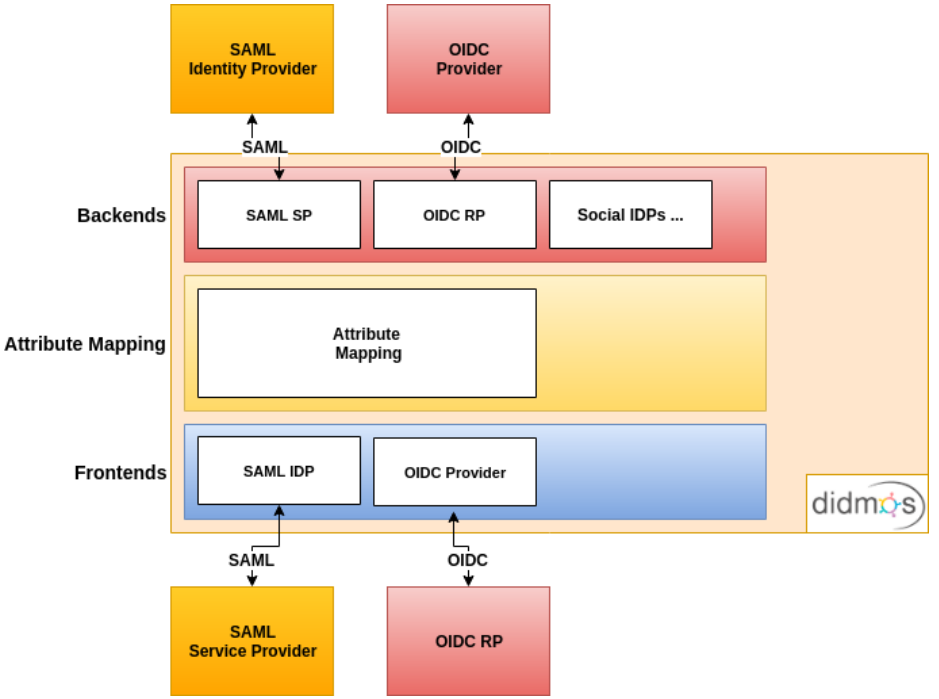
- Frontends zu den Diensten
  - SAML2 IDP
  - OIDC Provider
- Backends zur Authentifizierung
  - SAML2 SP
  - OIDC RP
  - Diverse “vorgefertigte” Module für ORCID, Github, Google, Facebook, …
  - Session-Management findet in dem jeweiligen Backend statt



Einführung in den Aufbau von Satosa

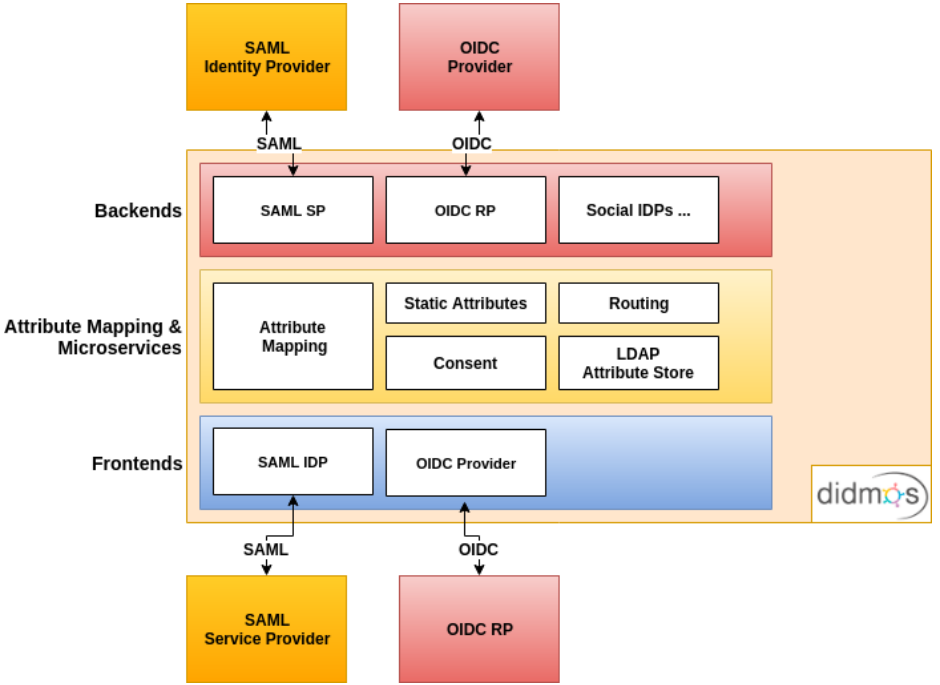
- Attribute-Mapping als Übersetzung zwischen den Protokollen

```
attributes:

  displayname:

    openid: [nickname]

    orcid: [name.credit-name]

    github: [login]

    saml: [displayName]

  mail:

    facebook: [email]

    linkedin: [email-address]

    orcid: [emails.str]

    openid: [email]

    saml: [email, emailAdress, mail]
```

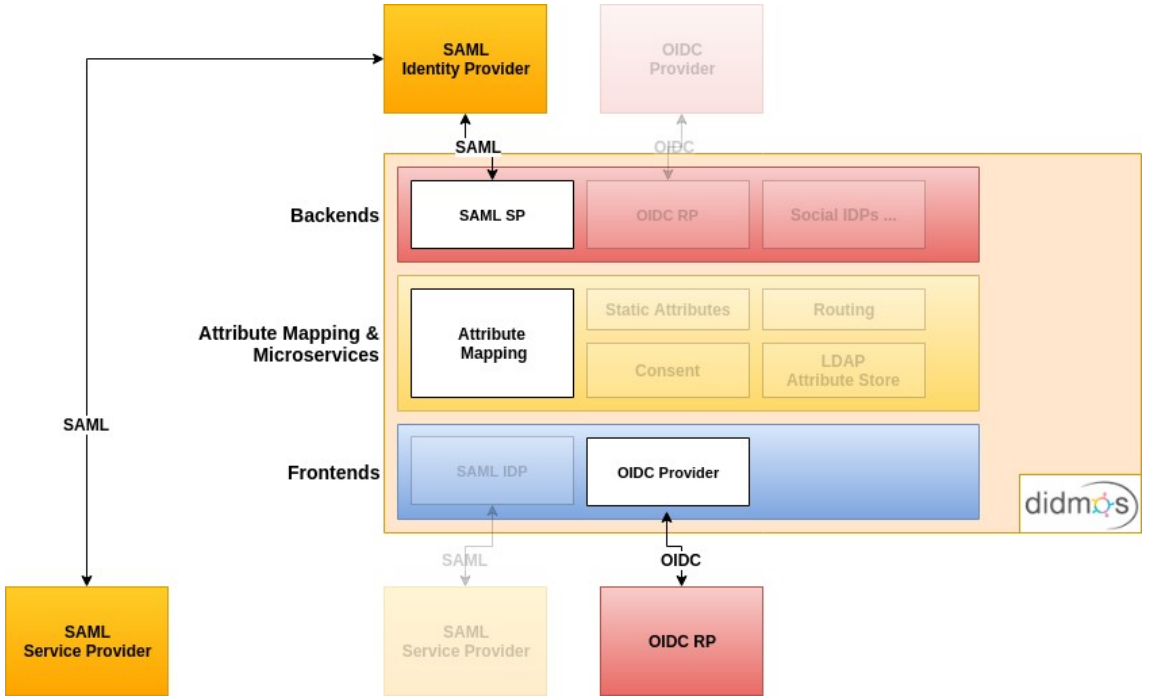


Einführung in den Aufbau von Satosa

- Microservices erlauben eine Manipulation von
  - Request
    - z.B. Routing (“welches Backend”)
  - Response
    - z.B. Attribute aus LDAP abrufen
    - Einbinden von externem Consent-Modul (API)

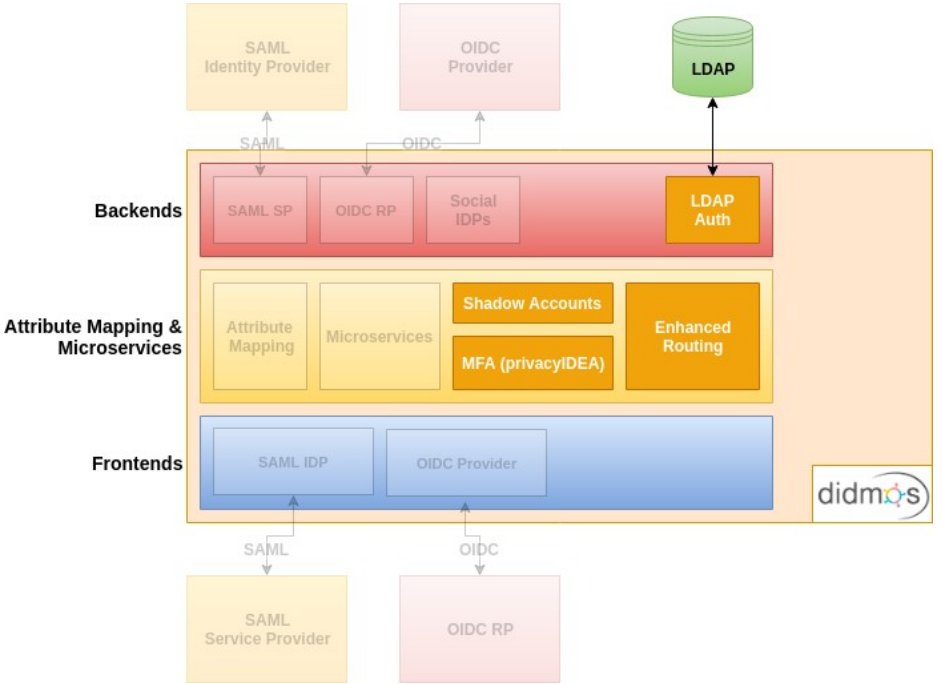


Usecase: Protokollübersetzung bei vorhandenem SAML IDP



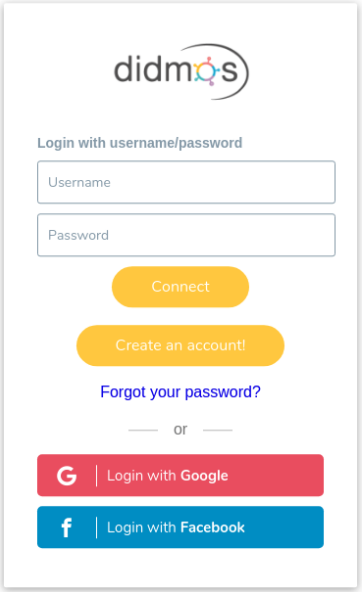
Erweiterung didmos Authenticator

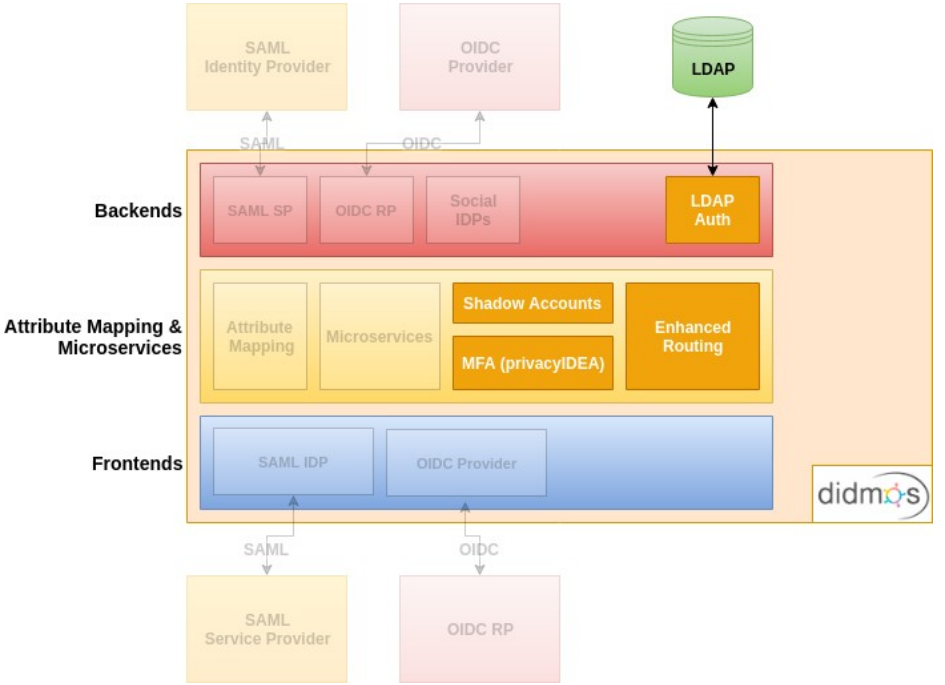
- Ziel: Ausbau zu einer vollwertigen und eigenständigen SSO-Lösung und Abstraktion der Konfiguration über Docker-Image
- LDAP Auth: Backend zur Authentifizierung gegen einen LDAP-Server
- Session-Management findet in dem LDAP-Auth-Backend statt (per verschlüsseltem Cookie)



Erweiterungen didmos Authenticator

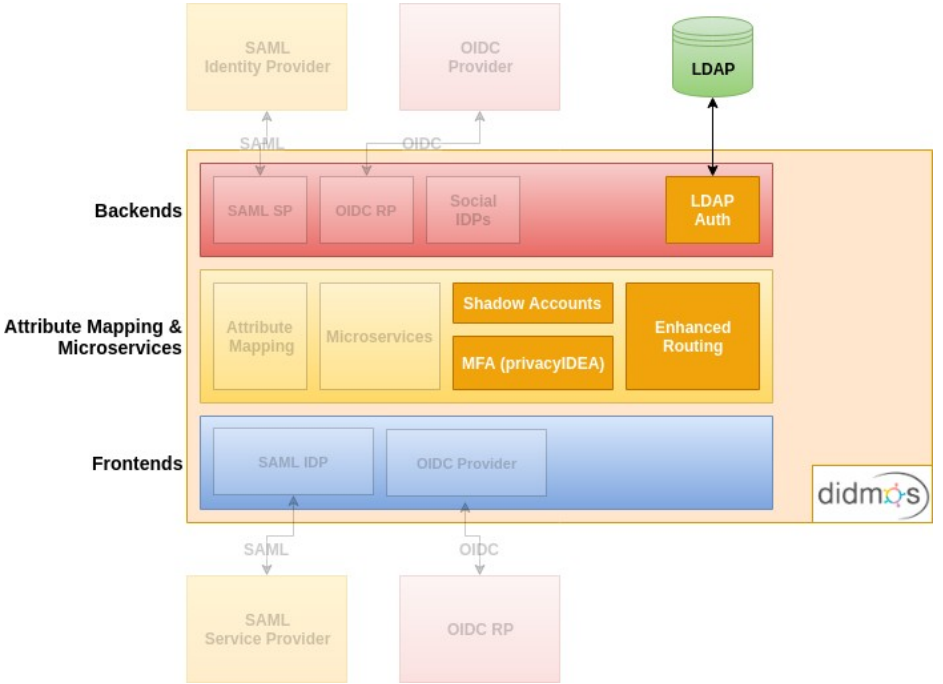
- Enhanced Routing: Modul zur Auswahl des Backends (dynamisch konfigurierbar je nach aktivierten Backends):



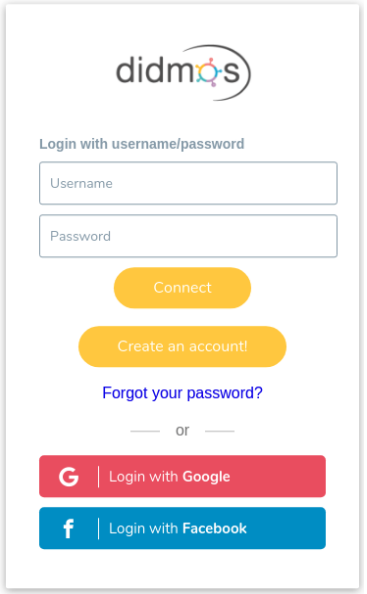


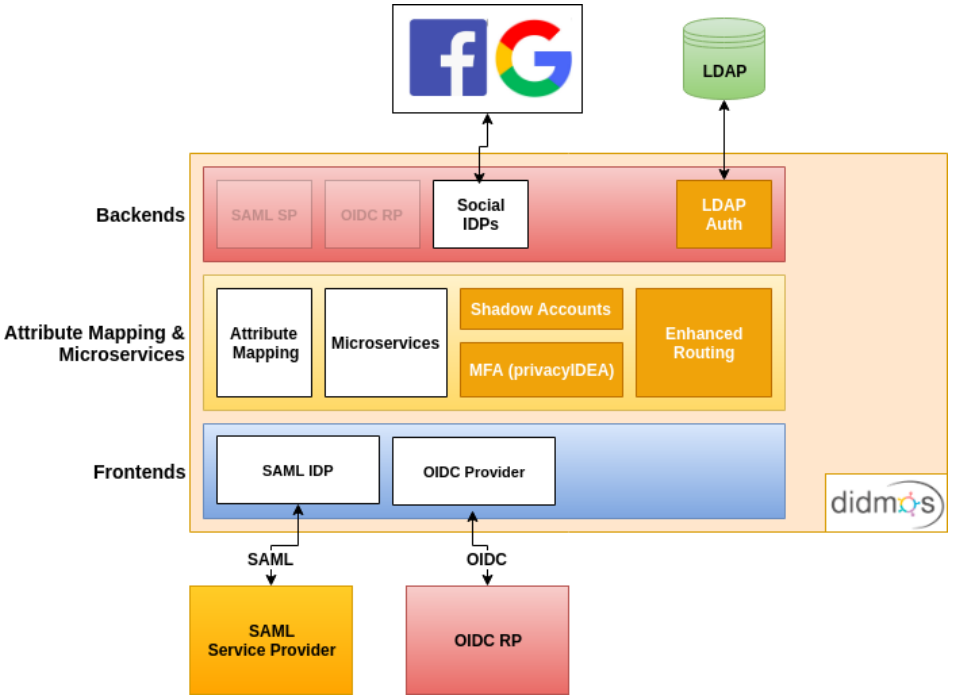
Erweiterungen didmos Authenticator

- Shadow Accounts: Nutzer, die sich über ein externes Backend authentifizieren, werden (wenn noch nicht vorhanden) im LDAP angelegt und können dann z.B. in eine lokale Rechte- oder Gruppenverwaltungs integriert werden



Usecase: Vollständige SSO-Lösung mit eigenem IDM und Social-IDP-Login





Szenario 3: Nutzung einer SSO-Lösung mit protokollübergreifendem Session-Handling

Beispiel: Shibboleth IDP + OIDC Extension

Einführung Shibboleth OIDC-Erweiterung

- Im Rahmen des GÉANT GN-42 Projekts wurde eine Java-Erweiterung für den Shibboleth IDP V3 erstellt
- Ende 2017 wurde eine erste Alpha veröffentlicht, Ende 2018 dann eine Beta und seit April 2019 gibt es eine v1.0.0
- Übernahme der Erweiterung in den IDP-Core mit V4 bzw. V5 geplant

DAASI International

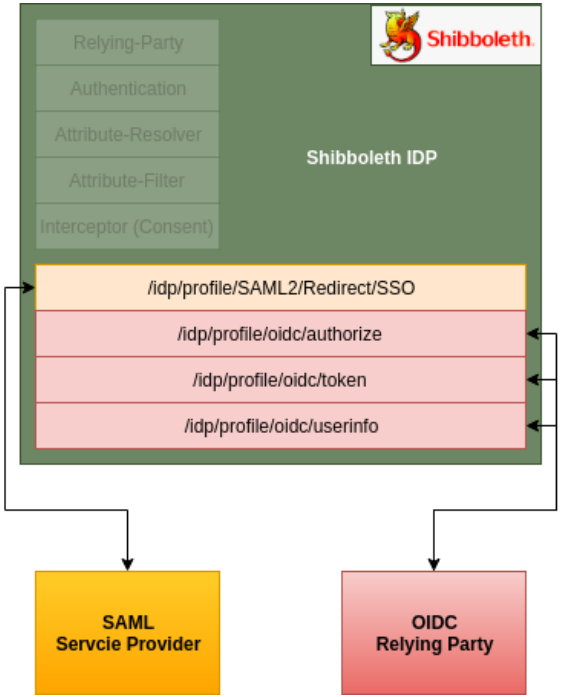
Einführung Shibboleth OIDC-Erweiterung

- Mittlerweile werden alle wichtigen OIDC Grants und Funktionen unterstützt
  - Authorization Code und Implicit Grant
  - Discovery & Dynamic Registration
  - Token Revocation (Access Token)
- Die Erweiterung kann leicht in eine vorhandene Shibboleth IDP V3-Installation integriert werden
- Er werden viele der vorhandenen Strukturen im IDP verwendet, was doppelte Konfiguration vermeidet

DAASI International

Aufbau der Erweiterung – Flows

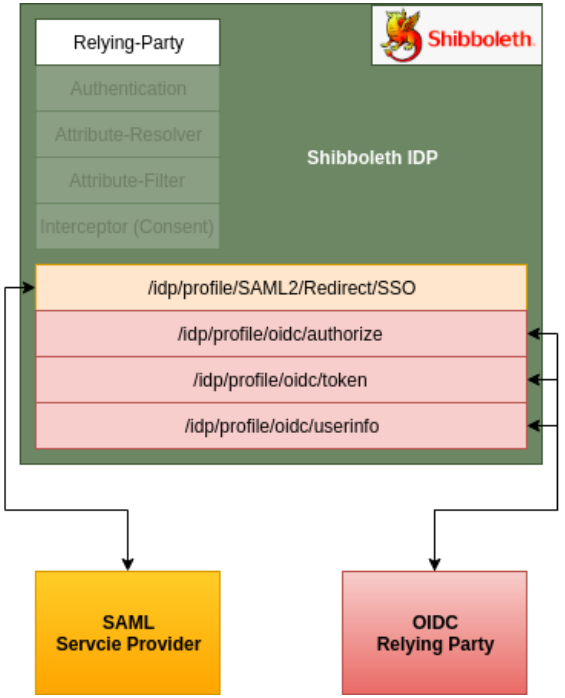
- Es werden diverse Flows definiert, die i.d.R. eigene Endpunkte hinzufügen:
  - oidc/authorize
  - oidc/token
  - oidc/userinfo
  - oidc/keyset
  - oidc/register
  - oauth2/revocation
- Diese Endpunkte können von OIDC-RPs angesprochen werden



Aufbau der Erweiterung - Relying-Party

- Die OIDC-Profiles (basierend auf den Flows) lassen sich analog zu den SAML-Profiles in die Relying-Party-Konfiguration aufnehmen
- Auch in Overrides (statt EntityID wird hier mit der ClientID gearbeitet)
- Für UnverifiedRelyingParty “OIDC.Registration” freigeben, um dynamische Registrierung zu erlauben

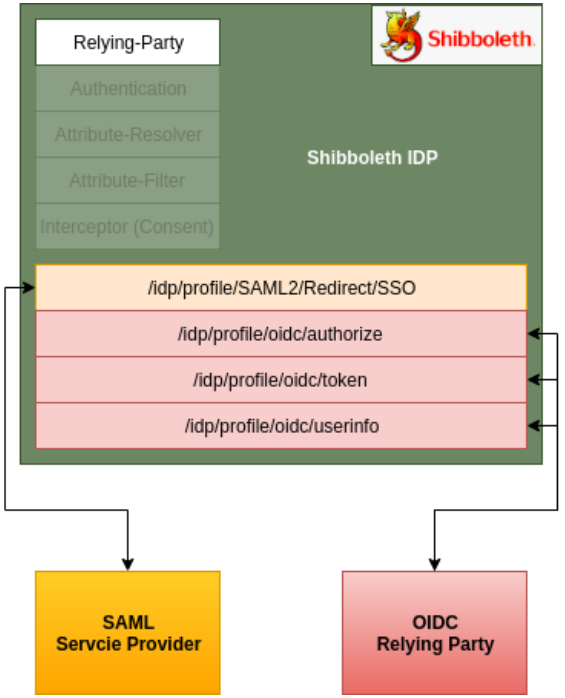
```
<bean id="shibboleth.DefaultRelyingParty"
      p:responderIdLookupStrategy-ref="profileResponderIdLookupFunction"
      parent="RelyingParty">
    <property name="profileConfigurations">
        <list>>
            <bean parent="SAML2.SSO" p:postAuthenticationFlows="attribute-release" />
            <ref bean="SAML2.ECP" />
            <ref bean="SAML2.Logout" />
            <bean parent="OIDC.SSO" p:postAuthenticationFlows="attribute-release" />
            <bean parent="OIDC.UserInfo"/>
            <bean parent="OAUTH2.Revocation"/>
        </list>
    </property>
</bean>
```



Aufbau der Erweiterung – Relying-Party cont’d

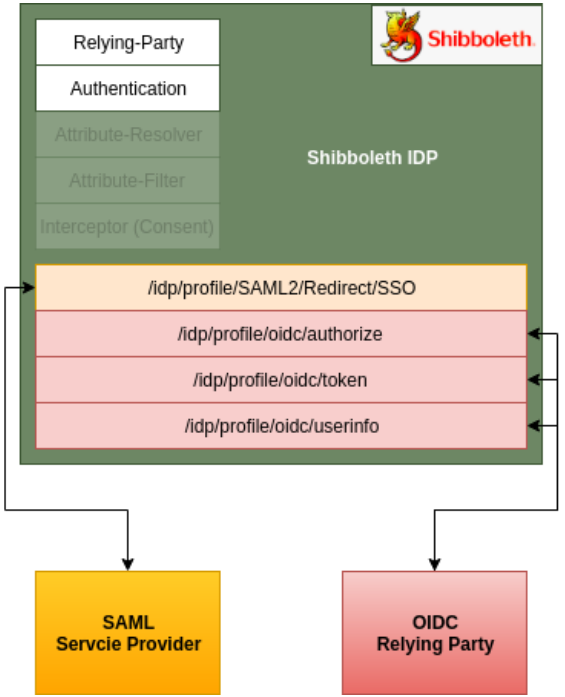
- “OIDC-Metadaten” lassen sich analog zu SAML-Metadaten über JSON-Dateien lokal verwalten...
- … oder in einem extra definierten Storage-Backend (für dynamische Registrierung notwendig)

```
[
  {
    "scope":"openid email",
    "redirect_uris":["https://oidc.example.org/redirect"],
    "client_id":"my_demo_client",
    "client_secret":"s3cr3t",
    "response_types":["code"],
    "grant_types":["authorization_code"]
  }
]
```



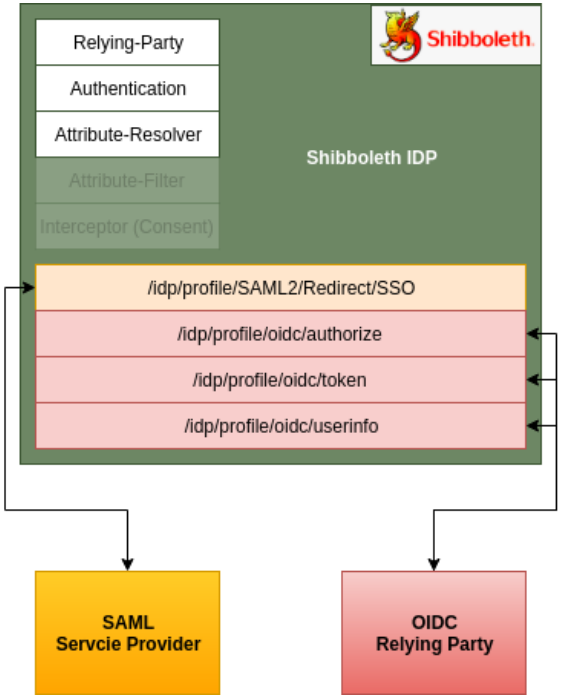
Aufbau der Erweiterung – Authentication

- Es werden die vorhandenen Authentifizierungs-Mechanismen verwendet (z.B. “Password”)
- Dabei wird auch auf das vorhandene Session-Management zurückgegriffen (und damit protokollübergreifendes SSO ermöglicht)



Aufbau der Erweiterung – Attribute-Resolver

- Im Resolver reicht es i.A. aus, für bestehende Attribute, weitere Encoder für die OIDC-Claims hinzuzufügen
- Für Identifier-Claims (z.B: “sub” als pairwise-Claim) können vorhandere Strategien zur Generieing von NameIDs genutzt werden

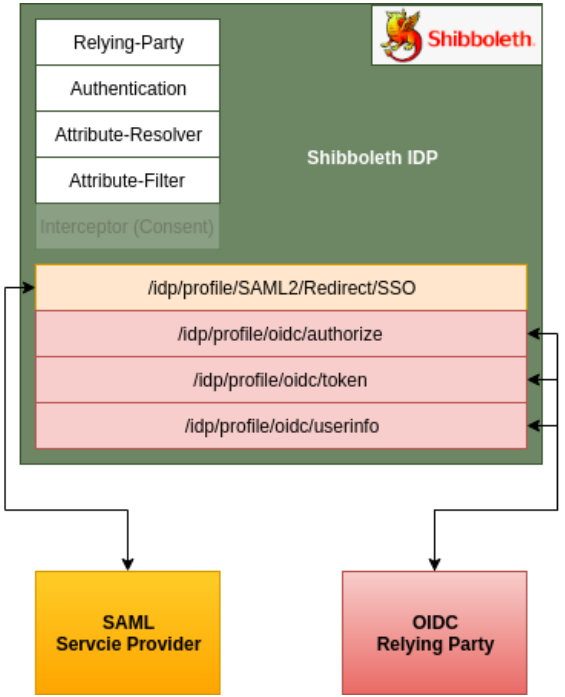



```
<AttributeDefinition id="uid" xsi:type="Simple">
    <InputDataConnector ref="myLDAP" attributeNames="uid"/>
    <AttributeEncoder xsi:type="SAML2String" name="urn:oid:0.9.2342.19200300.100.1.1"
                      friendlyName="uid" encodeType="false" />
    <AttributeEncoder xsi:type="oidcext:OIDCString" name="preferred_username" />
</AttributeDefinition>
```

Aufbau der Erweiterung – Attribute-Filter

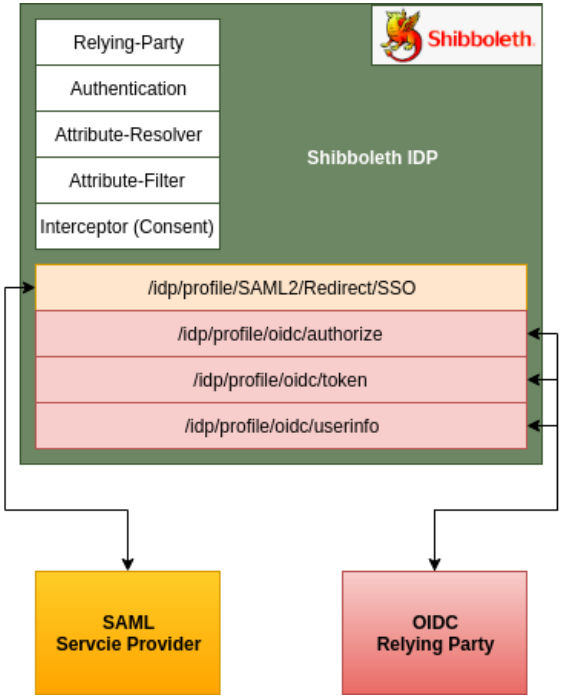
- Im Filter lassen sich Regelkombinationen für Requester (=ClientID) und angefordertem Scope formulieren
- So lassen sich auch Kombinationen von Scope und Claims abbilden

```
<AttributeFilterPolicy id="demo_client_email_scope">
    <PolicyRequirementRule xsi:type="AND">
        <Rule xsi:type="Requester" value="my_demo_client" />
        <Rule xsi:type="oidcext:OIDCScope" value="email" />
    </PolicyRequirementRule>
    <AttributeRule attributeID="mail">
        <PermitValueRule xsi:type="ANY" />
    </AttributeRule>
</AttributeFilterPolicy>
```



Aufbau der Erweiterung – Interceptors

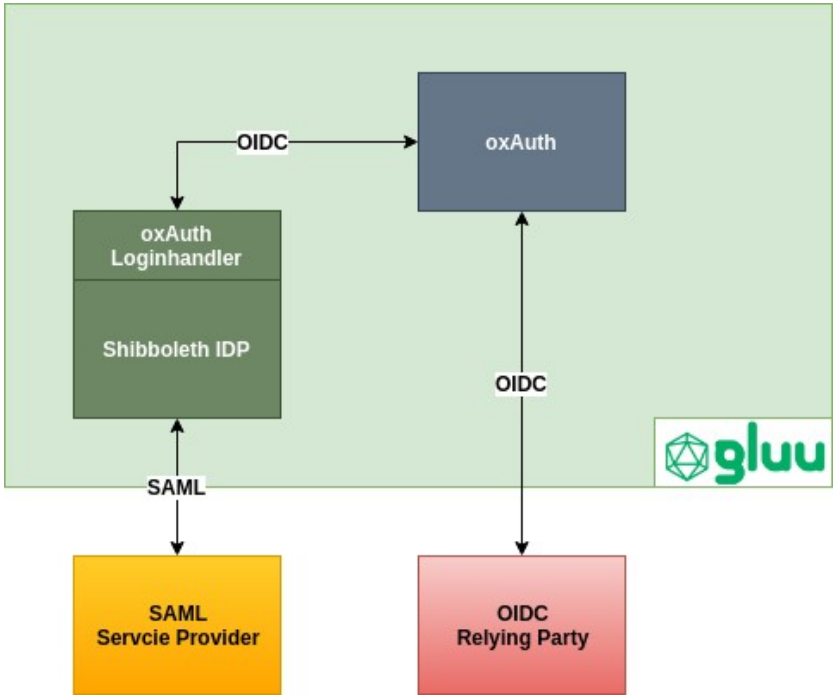
- Vorhandene Interceptors (z.B. “attribute-release”) lassen sich weiternutzen und in der Relying-Party-Konfiguration einbinden




```
<bean id="shibboleth.DefaultRelyingParty"
      p:responderIdLookupStrategy-ref="profileResponderIdLookupFunction"
      parent="RelyingParty">
    <property name="profileConfigurations">
        <list>>
            <bean parent="SAML2.SSO" p:postAuthenticationFlows="attribute-release" />
            <ref bean="SAML2.ECP" />
            <ref bean="SAML2.Logout" />
            <bean parent="OIDC.SSO" p:postAuthenticationFlows="attribute-release" />
            <bean parent="OIDC.UserInfo"/>
            <bean parent="OAUTH2.Revocation"/>
        </list>
    </property>
</bean>
```

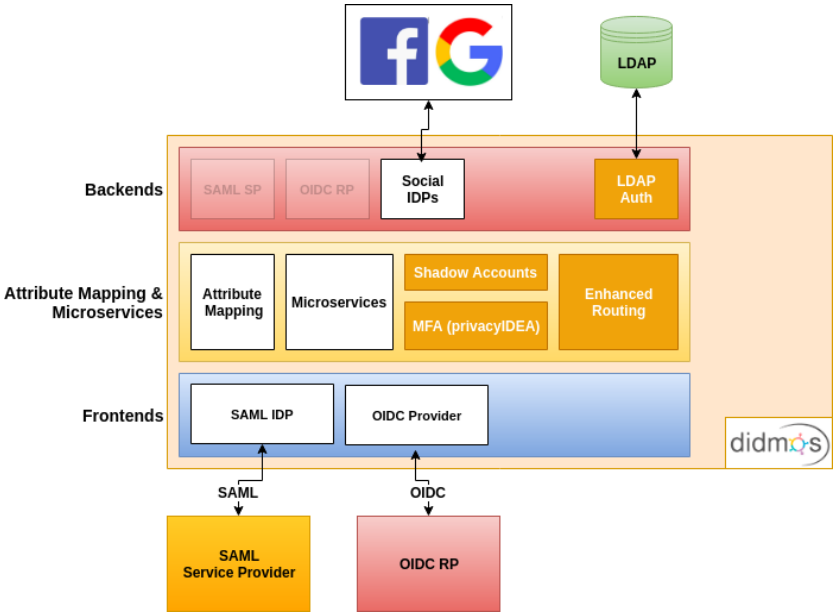
Recap: Gluu

- Im Gluu-Szenario wird der Login über den SAML-IDP einfach per OIDC-Loginhandler an oxAuth weitergereicht
- SAML SPs nutzen den Shibboleth IDP, OIDC RPs direkt oxAuth
- Session-Management ist nur an einer Stelle nötig: in oxAuth



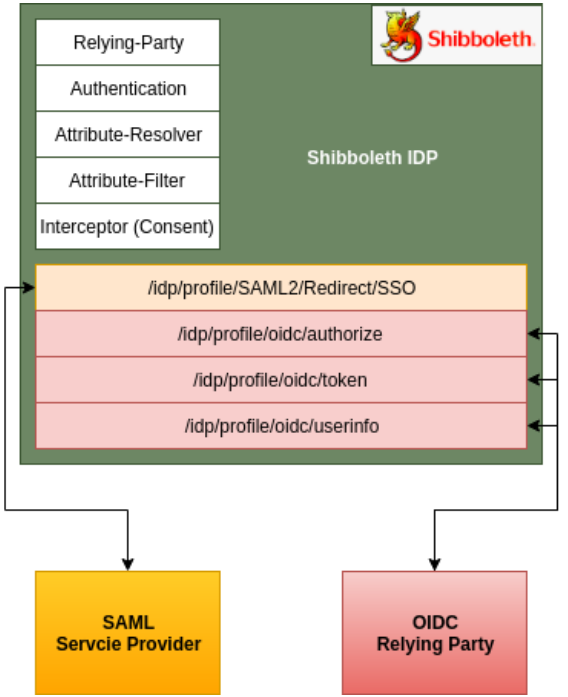
Recap: Satosa & didmos Authenticator

- didmos Authenticator erweitert den Proxy Satosa um Features einer vollständigen SSO-Lösung, jedoch mit Fokus auf die Proxy-Anwendungsfälle
- Für jedes Protokoll wird ein eigenes Frontend eingesetzt
- Session-Management findet direkt im jeweiligen Backend statt



Recap: Shibboleth IDP + OIDC-Extension

- Die Erweiterung fügt OIDC-Profiles und -Endpunkte im Shibboleth IDP hinzu
- Alle weiteren Features des IDP können integriert werden
- Es wird auf das bestehende Session-Management zurückgegriffen



Vergleich der Lösungen

• Beste OIDC/OAuth2-Fähigkeiten der Lösungen • Single-Logout über beide Protokolle möglich	• Modulare Lösung mit hoher Erweiterbarkeit • Single-Logout teilweise möglich • Proxy-Funktionalität	• Integration in (oft) bereits existierende Software-Lösung • Weitgehend weiterverwendung der vorhandenen Konfiguration
• Komplexität im Vergleich zu den anderen Produkten hoch	• Limitierte OAuth2-Fähigkeiten • Doku noch nicht auf dem Stand der anderen Lösungen	• Limitierte OAuth2-Fähigkeiten • Derzeit kein Single-Logout möglich • Derzeit nur als Erweiterung

DAASI International

Vielen Dank für Ihre Aufmerksamkeit.

David Hübner

DAASI International
www.daasi.de

E-Mail: david.huebner@daasi.de