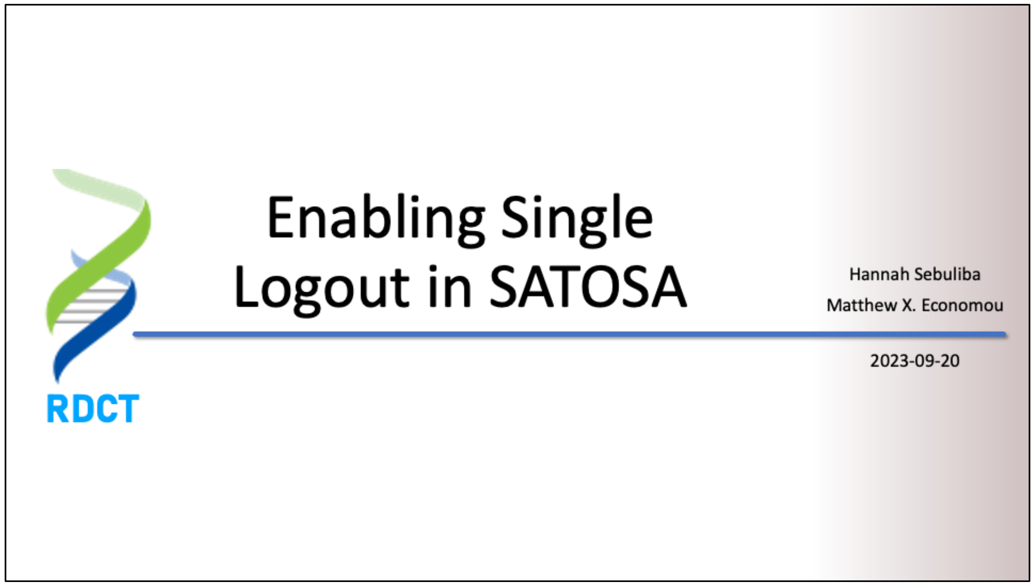


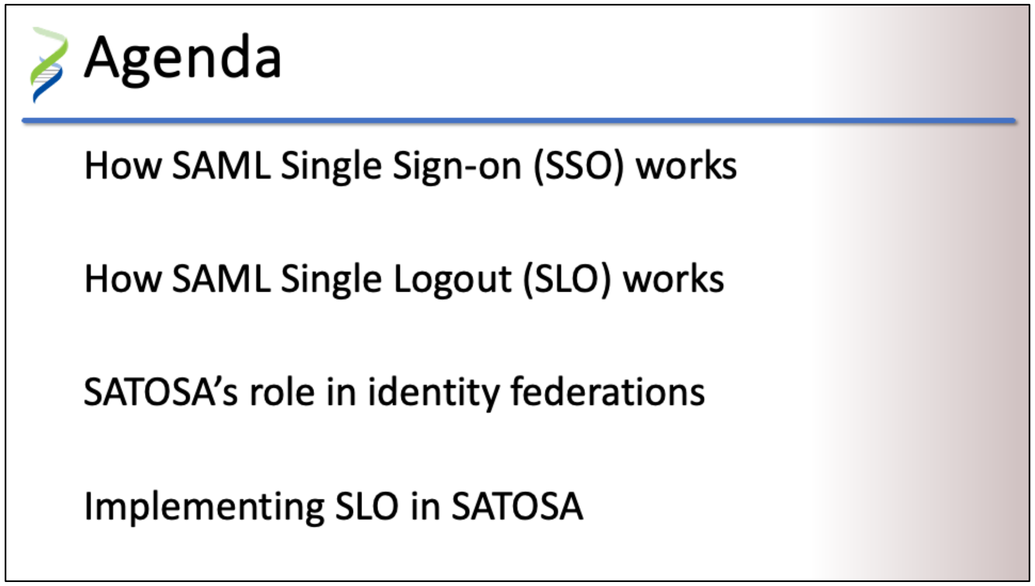
Enabling Single Logout in SATOSA

RDCT

Hannah Sebuliba
Matthew X. Economou

2023-09-20

Good morning. I'm Matthew Economou, Vice President of Engineering for Research Data and Communication Technologies. My colleague, Hannah Sebuliba, who developed the Single Logout code for SATOSA, is unfortunately unable to attend. I am presenting in her stead.



Agenda

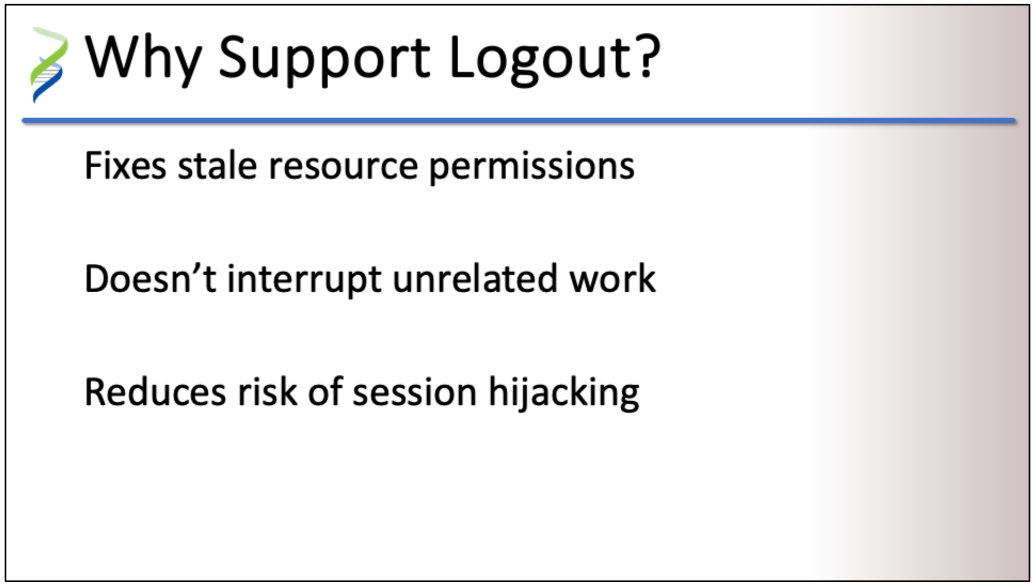
How SAML Single Sign-on (SSO) works

How SAML Single Logout (SLO) works

SATOSA's role in identity federations

Implementing SLO in SATOSA

We have four major topics today. First, we'll quickly review how SAML Web Single Sign-on works and, second, how SAML Single Logout extends the SSO profile. Third, we'll go over SATOSA's role in identity federations before, finally, digging into our implementation of SLO in SATOSA.



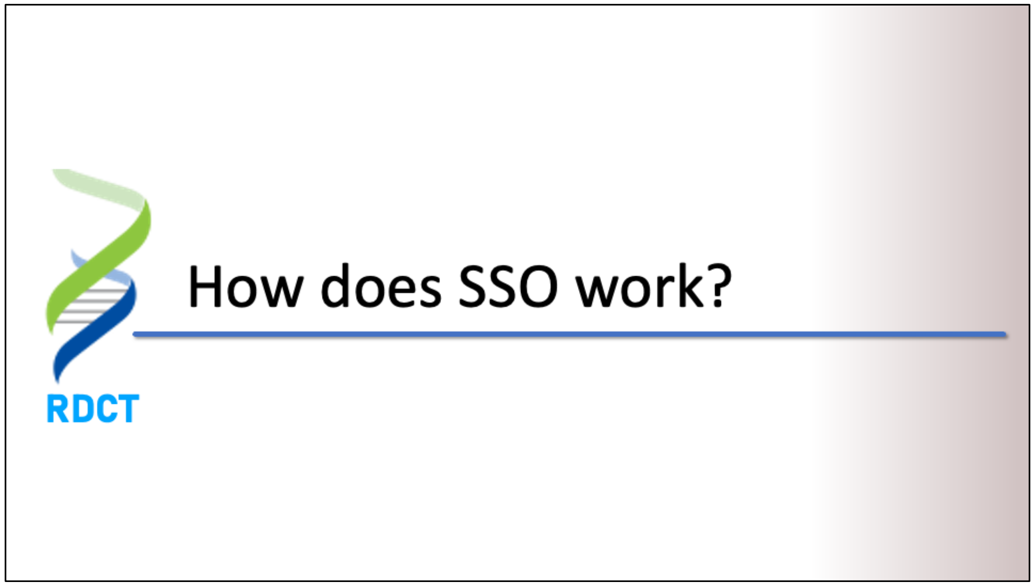
Why Support Logout?

Fixes stale resource permissions

Doesn’t interrupt unrelated work

Reduces risk of session hijacking

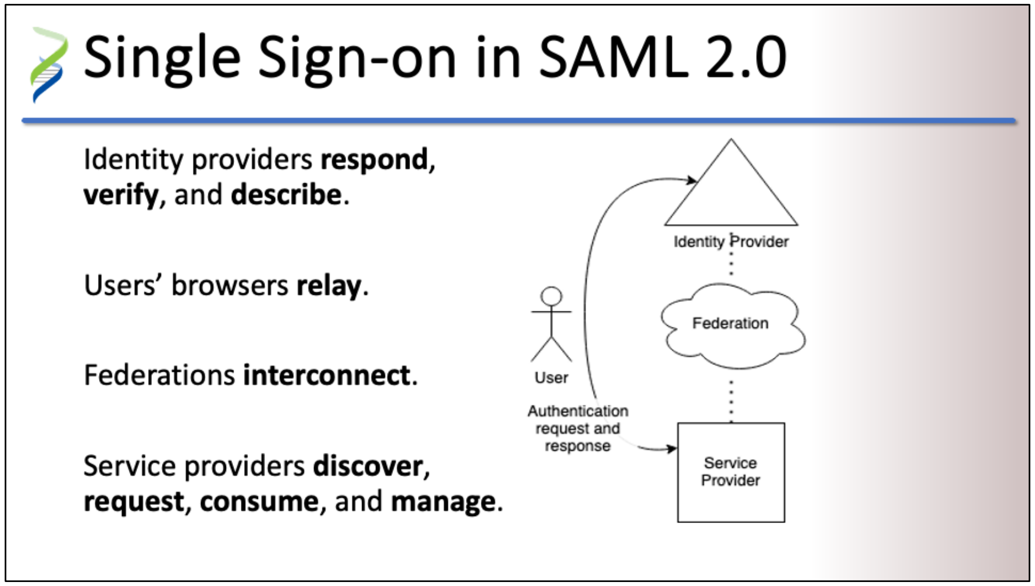
We have a few reasons for wanting a functional logout button. A big one is technical support. Having a working logout/login workflow makes it easy for users to fix stale resource permissions or rights assignments. Not everyone can use temporary browser sessions, e.g., one of our clients explicitly disables Chrome’s InPrivate browsing feature, and many users do not know how to selectively clear browser cookies, which can interrupt other work should they wipe their entire browsing history. Another angle is information security. We want users to be able to explicitly terminate their login sessions as this reduces the risk of successful session hijacking attacks.



How does SSO work?

RDCT

But before we talk about what "logging out" means in an identity federation, let's quickly review what it means to "log in" with SAML.



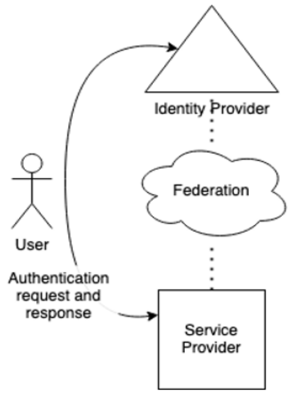
Single Sign-on in SAML 2.0

Identity providers **respond**, **verify**, and **describe**.

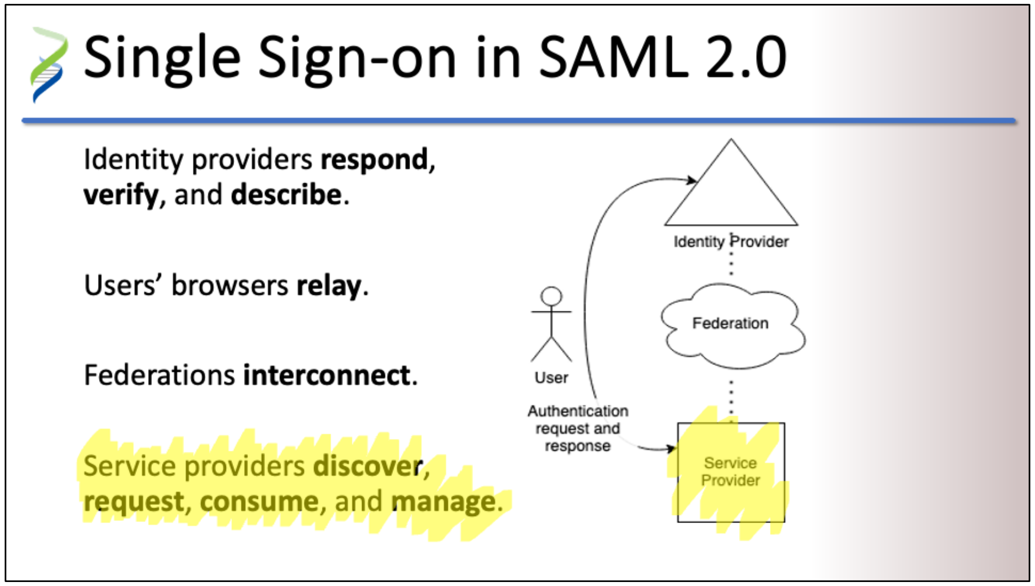
Users' browsers **relay**.

Federations **interconnect**.

Service providers **discover**, **request**, **consume**, and **manage**.



The SAML 2.0 Web Browser SSO profile connects service provider and identity provider logins.



Single Sign-on in SAML 2.0

Identity providers **respond**, **verify**, and **describe**.

Users' browsers **relay**.

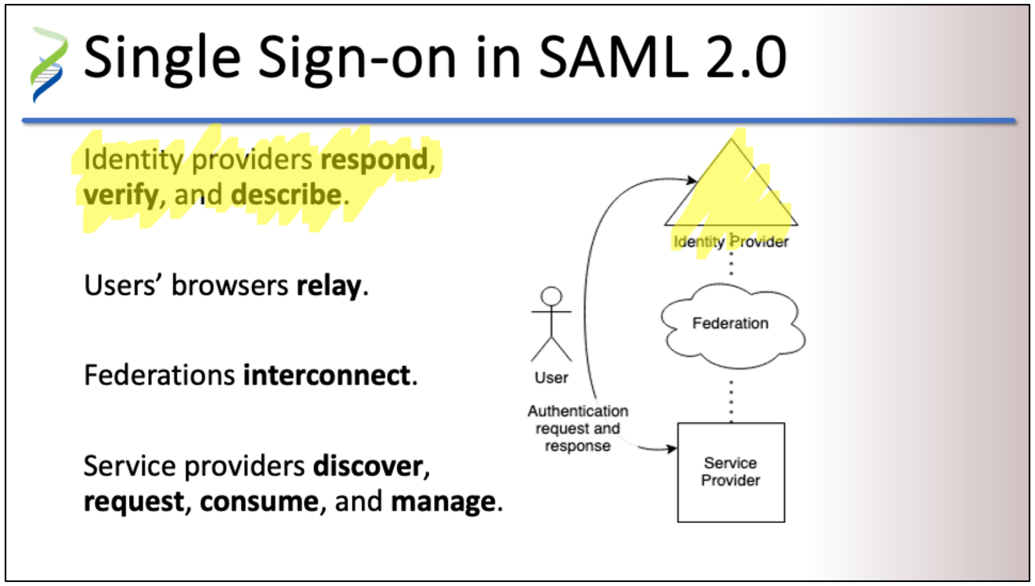
Federations **interconnect**.

Service providers **discover**, **request**, **consume**, and **manage**.

Identity Provider
User
Federation
Authentication request and response
Service Provider

A service provider (or relying party) is any web app that depends on separate web service to authenticate—and sometimes to authorize—the web app's users. The SP must do four things:

- **discover** where a user is from
- **request** user authentication by the selected identity provider
- **consume** user information released by the IdP, e.g., unique identifiers, contact information, group memberships, entitlements
- **manage** the user's session while controlling access to app resources and functionality



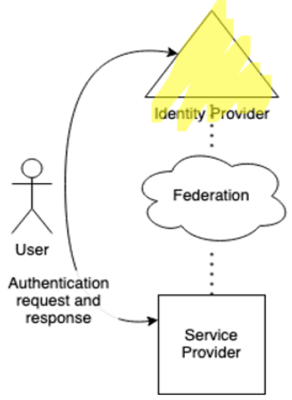
Single Sign-on in SAML 2.0

Identity providers **respond**, **verify**, and **describe**.

Users' browsers **relay**.

Federations **interconnect**.

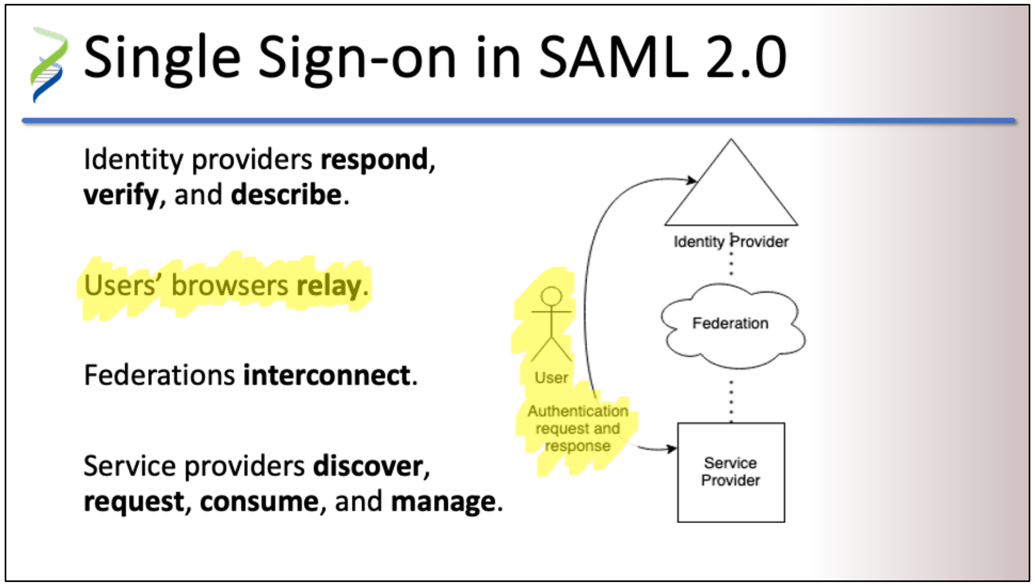
Service providers **discover**, **request**, **consume**, and **manage**.



An identity provider (IdP) is a web service that authenticates users who are accredited by its operator. The IdP does three things:

- **responds** to authentication requests
- **verifies** the user's credentials
- **describes** the user to the SP, e.g., affiliations, entitlements, identifiers, assurances

(An IdP can also **initiate** the login process on its own in some cases, but that isn't part of today's discussion.)



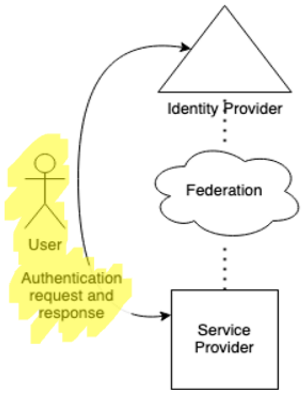
Single Sign-on in SAML 2.0

Identity providers **respond**, **verify**, and **describe**.

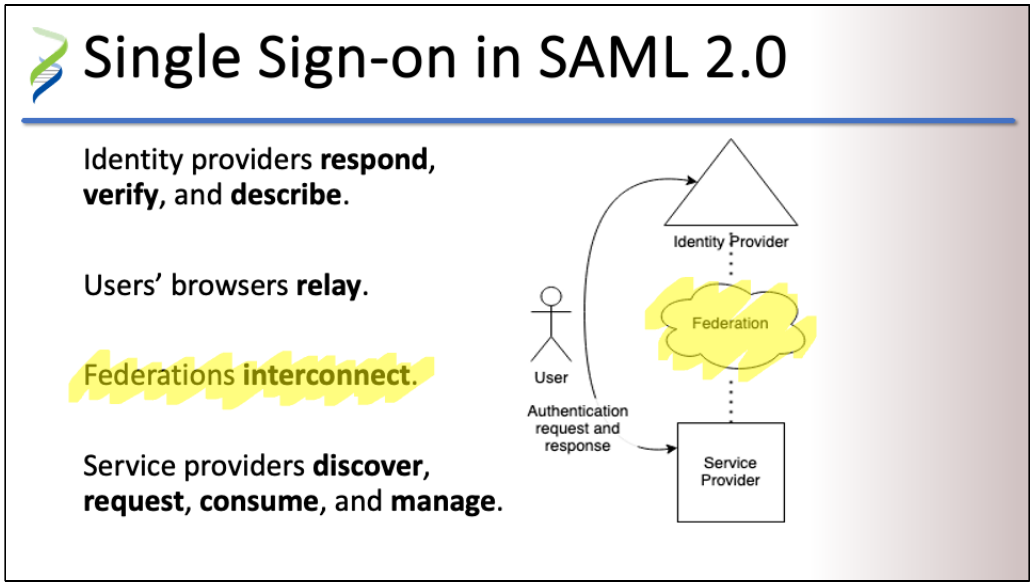
Users' browsers **relay**.

Federations **interconnect**.

Service providers **discover**, **request**, **consume**, and **manage**.



In describing a user, the IdP will retrieve or synthesize user information from one or more data sources, such as an enterprise directory service. This information is returned to the SP in a SAML <Response> message. All SAML messages—requests and responses—are relayed by the user's web browser (front-channel presentation).



Single Sign-on in SAML 2.0

Identity providers **respond**, **verify**, and **describe**.

Users' browsers **relay**.

Federations **interconnect**.

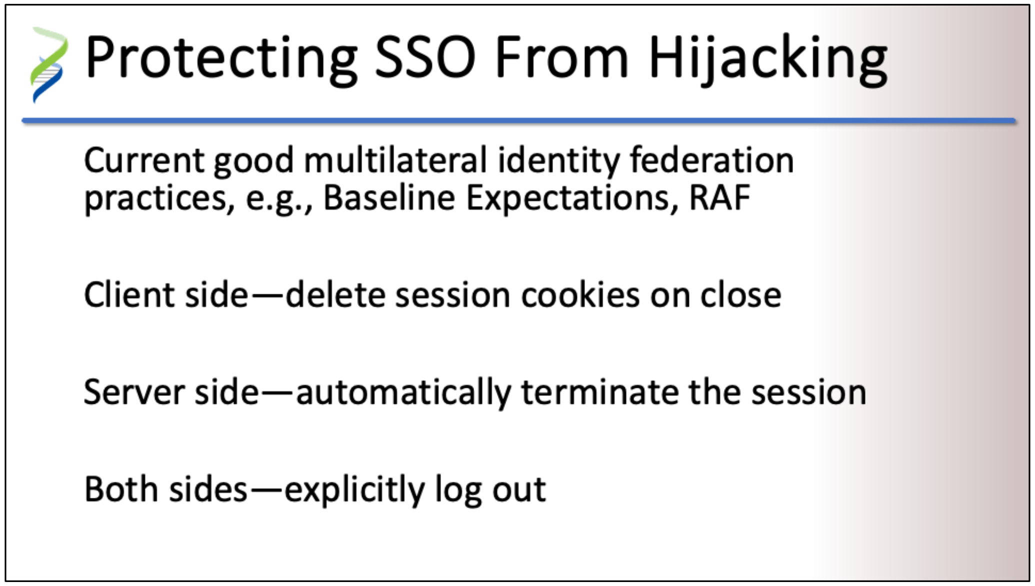
Service providers **discover**, **request**, **consume**, and **manage**.

Identity Provider
Federation
User
Authentication request and response
Service Provider

At scale, SPs and IdPs have to solve the identity and access coordination problem:

- If you operate a service provider, you want many people from many institutions to use your app.
- If you operate an identity provider, you want your users to be able to access a wide variety of useful services.

An identity federation like InCommon is how service providers and identity providers find and trust one another.



Protecting SSO From Hijacking

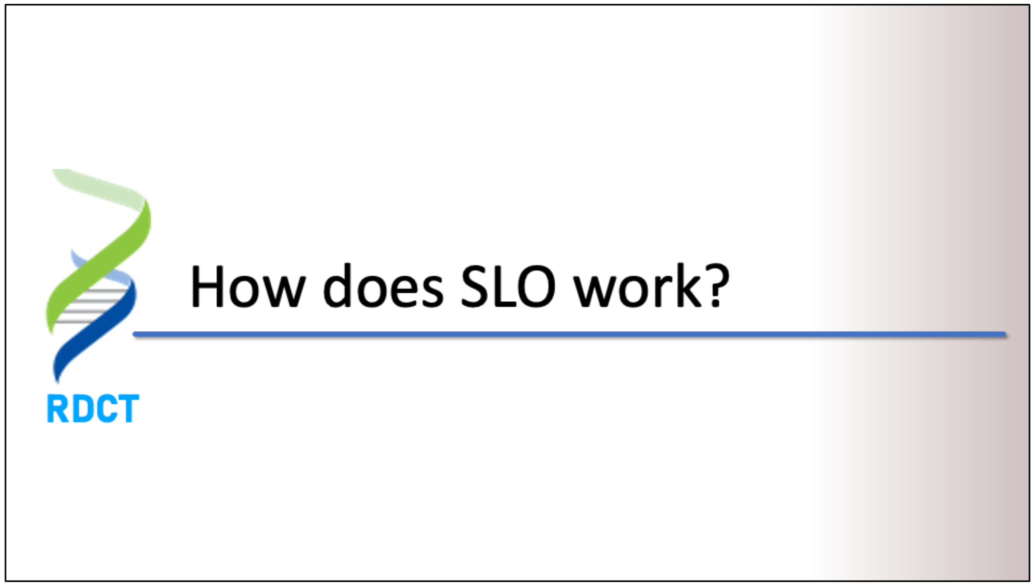
Current good multilateral identity federation practices, e.g., Baseline Expectations, RAF

Client side—delete session cookies on close

Server side—automatically terminate the session

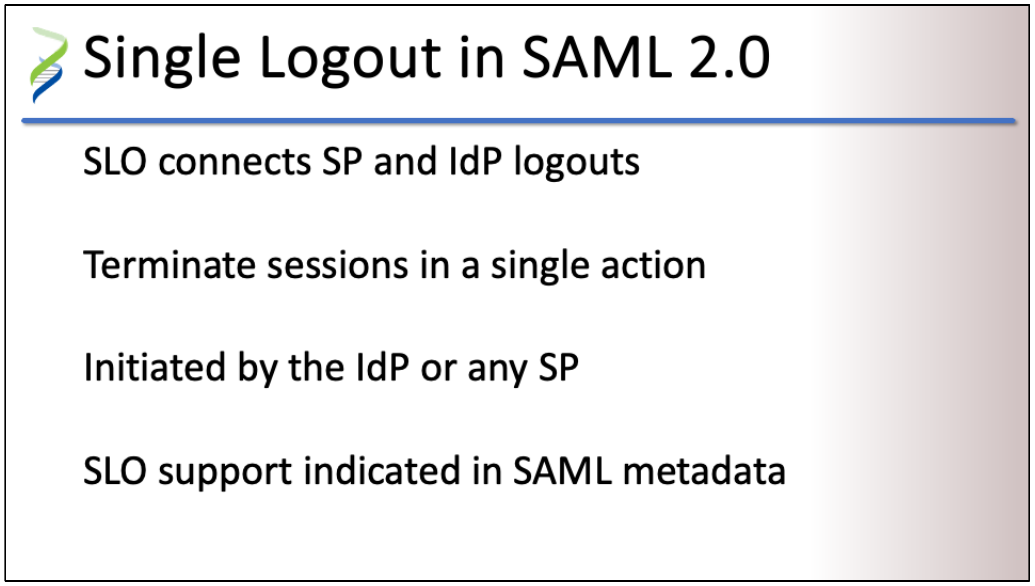
Both sides—explicitly log out

Protecting the confidentiality, integrity, and availability of Single Sign-on means following current good multilateral identity federation practices, which are a variety of entity categories, frameworks, profiles, and services—some of which you'll learn about in other talks here, such as InCommon Baseline Expectations or the REFEDS Assurance Framework. In this talk, we're focusing on protecting sessions from hijacking. The simplest session hijacking mitigation relies on the browser just deleting session cookies when the user closes a tab or a window. Hackers can't exfiltrate data that doesn't exist, but this still leaves the session active server-side. And technically, the browser only deletes references to the session data in memory. That data might still be recoverable via crash dumps or other debugging tools. (This happened to Microsoft recently.) The server can automatically end the session after a period of inactivity, but that still leaves one vulnerable until the session actually times out server-side. A way to further mitigate this risk of session hijacking is for the user to explicitly ask their web applications and identity provider to log out because then the session is deleted (or invalidated) on both the client side and the server side, simultaneously. This is why Single Logout is important—it implements “explicitly logging out” in the context of a SAML federation.



How does SLO work?

RDCT



Single Logout in SAML 2.0

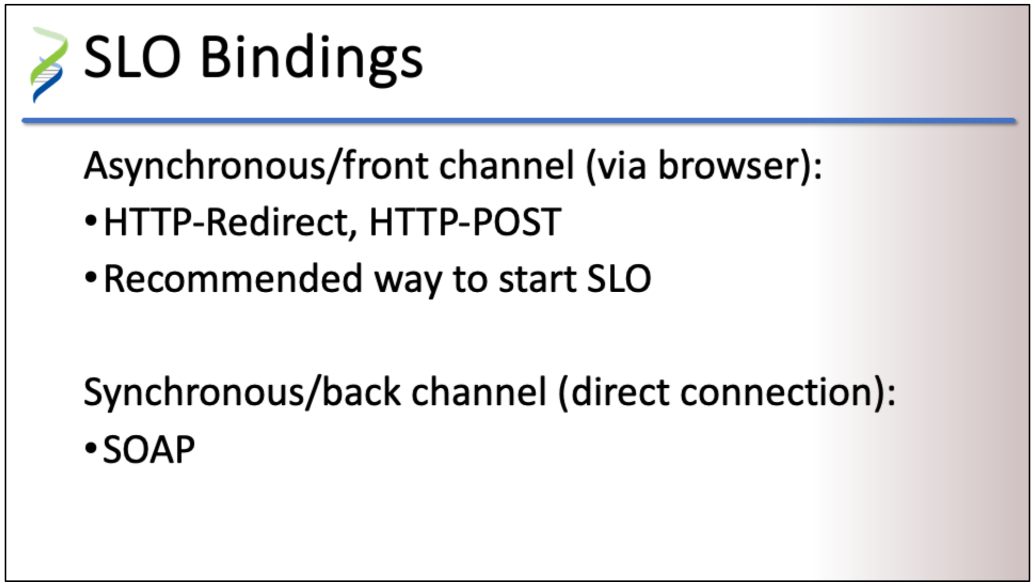
SLO connects SP and IdP logouts

Terminate sessions in a single action

Initiated by the IdP or any SP

SLO support indicated in SAML metadata

Just like how the SAML 2.0 Web Browser SSO profile connects service provider and identity provider logins, the Single Logout profile links service provider and identity provider logouts. With SLO in place, users can choose to terminate with a single action their sessions on all web apps accessed in this browser—that is, all web apps that support SLO. Single Logout is bi-directional. The logout process can be started by any entity with an active session, SP or IdP. Because of the decentralized nature of identity federations—remember, SPs and IdPs communicate indirectly through users' web browsers—whether a given entity supports SLO must be reflected in their federation metadata. Note that automatic session termination, e.g., an inactivity timeout, does not trigger the Single Logout flow.



SLO Bindings

Asynchronous/front channel (via browser):

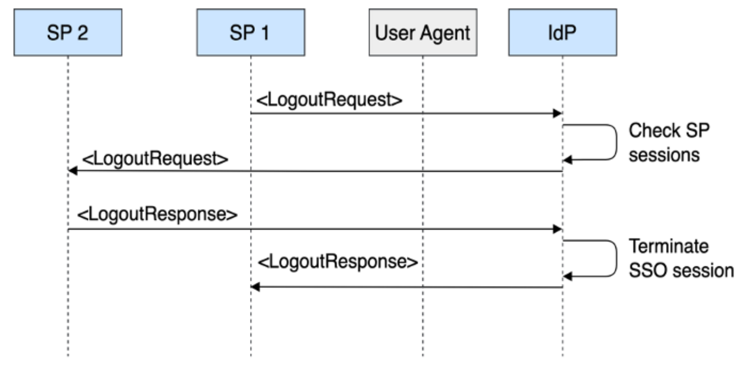
- HTTP-Redirect, HTTP-POST
- Recommended way to start SLO

Synchronous/back channel (direct connection):

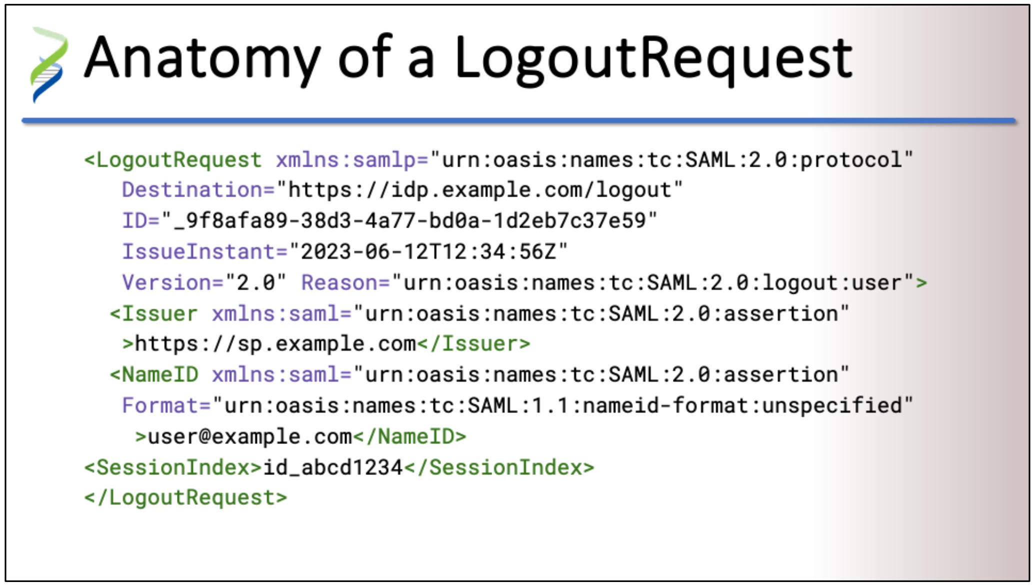
- SOAP

Single Logout works like Single Sign-on, with logout requests and responses being exchanged asynchronously, via the user's browser, or synchronously, with IdPs and SPs communicating directly via the Simple Object Access Protocol (SOAP). Front-channel presentation is generally regarded as being less complicated to implement and quicker to start the logout process. However, it can be vulnerable to cross-site scripting (XSS) or cross-site request forgery (CSRF) attacks, requiring careful mitigation by implementers. Back-channel presentation of the logout request and response isn't vulnerable to XSS or CSRF attacks, but it's widely regarded as more difficult to implement and deploy.

SP-initiated Logout



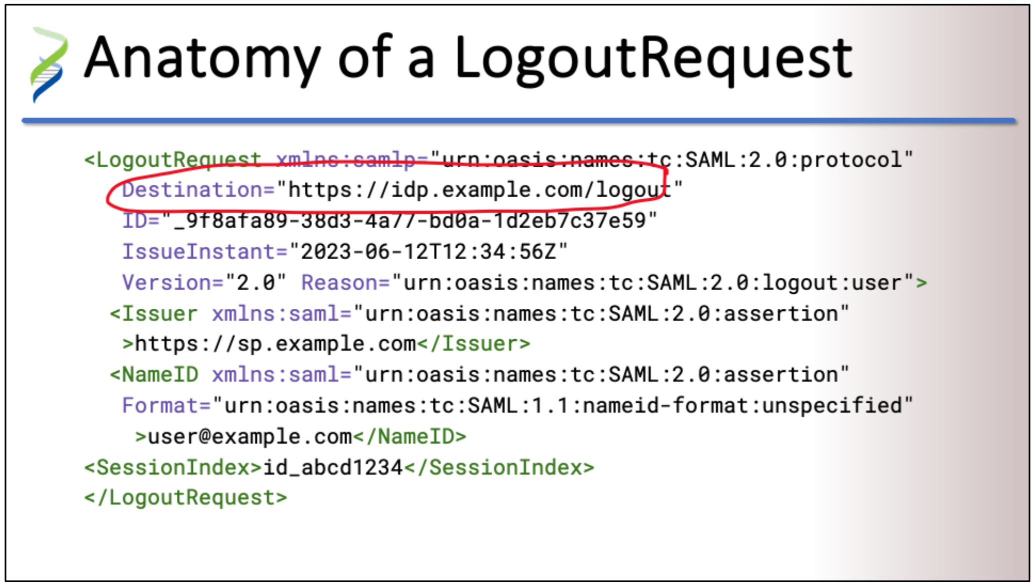
Here, we're focusing on SP-initiated front-channel logouts. Where the SSO profile uses <AuthenticationRequest> and <Response> messages to perform federated logins, the SLO profile uses <LogoutRequest> and <LogoutResponse> messages. Because only the IdP knows everywhere the user has logged in, the IdP is responsible for relaying logout requests from the initiating service provider to all the other SPs.



Anatomy of a LogoutRequest

```
<LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
   Destination="https://idp.example.com/logout"
   ID="_9f8afa89-38d3-4a77-bd0a-1d2eb7c37e59"
   IssueInstant="2023-06-12T12:34:56Z"
   Version="2.0" Reason="urn:oasis:names:tc:SAML:2.0:logout:user">
  <Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
   >https://sp.example.com</Issuer>
  <NameID xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
   Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"
    >user@example.com</NameID>
<SessionIndex>id_abcd1234</SessionIndex>
</LogoutRequest>
```

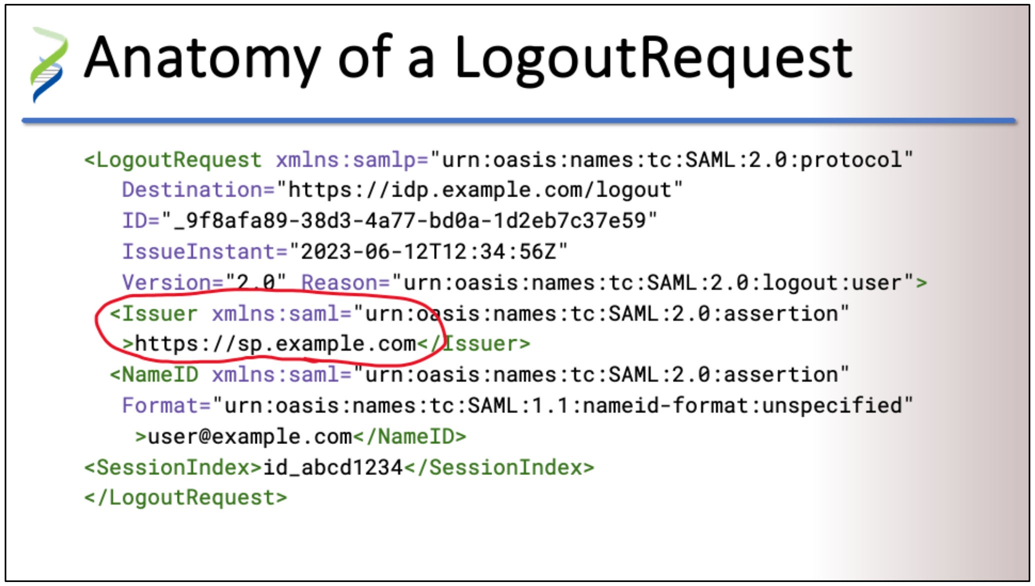
A <LogoutRequest> message has several parts, but let me draw your attention to these four:



Anatomy of a LogoutRequest

```
<LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
   Destination="https://idp.example.com/logout"
   ID="_9f8afa89-38d3-4a77-bd0a-1d2eb7c37e59"
   IssueInstant="2023-06-12T12:34:56Z"
   Version="2.0" Reason="urn:oasis:names:tc:SAML:2.0:logout:user">
  <Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
   >https://sp.example.com</Issuer>
  <NameID xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
   Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"
    >user@example.com</NameID>
<SessionIndex>id_abcd1234</SessionIndex>
</LogoutRequest>
```

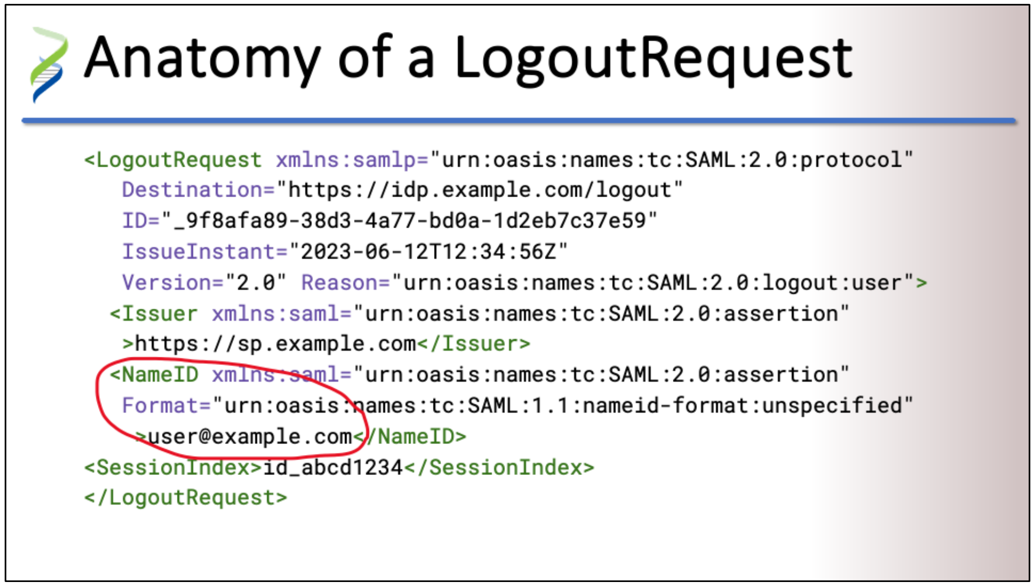
1. Destination, restating where the LogoutRequest was sent by the browser



Anatomy of a LogoutRequest

```
<LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
   Destination="https://idp.example.com/logout"
   ID="_9f8afa89-38d3-4a77-bd0a-1d2eb7c37e59"
   IssueInstant="2023-06-12T12:34:56Z"
   Version="2.0" Reason="urn:oasis:names:tc:SAML:2.0:logout:user">
  <Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
   >https://sp.example.com</Issuer>
  <NameID xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
   Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"
    >user@example.com</NameID>
<SessionIndex>id_abcd1234</SessionIndex>
</LogoutRequest>
```

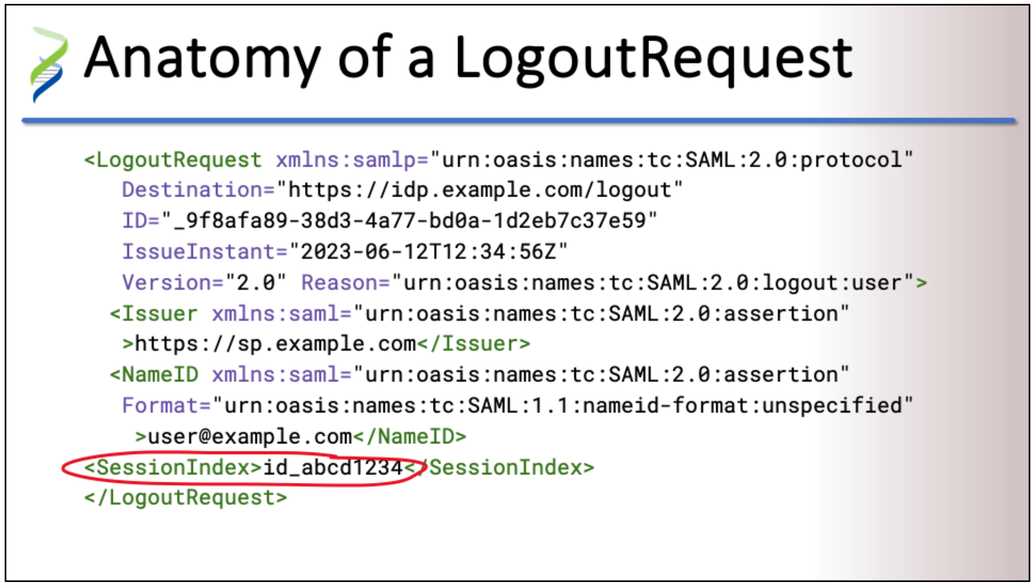
2. <Issuer>, the entity ID of the SP or IdP initiating SLO



Anatomy of a LogoutRequest

```
<LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
   Destination="https://idp.example.com/logout"
   ID="_9f8afa89-38d3-4a77-bd0a-1d2eb7c37e59"
   IssueInstant="2023-06-12T12:34:56Z"
   Version="2.0" Reason="urn:oasis:names:tc:SAML:2.0:logout:user">
  <Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
   >https://sp.example.com</Issuer>
  <NameID xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
   Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"
    >user@example.com</NameID>
<SessionIndex>id_abcd1234</SessionIndex>
</LogoutRequest>
```

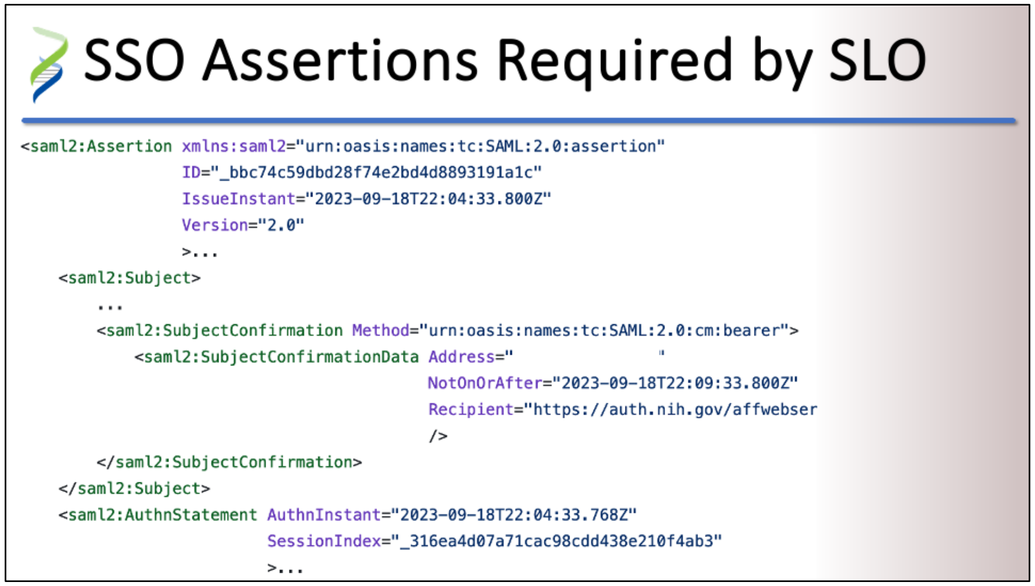
3. <NameID>, specifying which user initiated the logout



Anatomy of a LogoutRequest

```
<LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
   Destination="https://idp.example.com/logout"
   ID="_9f8afa89-38d3-4a77-bd0a-1d2eb7c37e59"
   IssueInstant="2023-06-12T12:34:56Z"
   Version="2.0" Reason="urn:oasis:names:tc:SAML:2.0:logout:user">
  <Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
   >https://sp.example.com</Issuer>
  <NameID xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
   Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"
    >user@example.com</NameID>
<SessionIndex>id_abcd1234</SessionIndex>
</LogoutRequest>
```

4. SessionIndex, which uniquely identifies the SSO session for this browser at the IdP



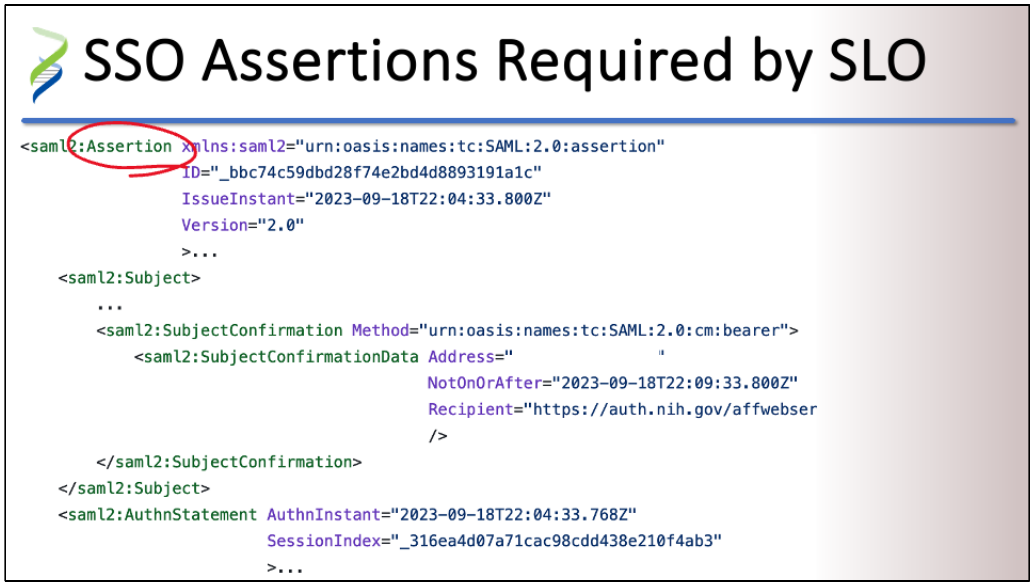
SSO Assertions Required by SLO

```
<saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
                 ID="_bbc74c59dbd28f74e2bd4d8893191a1c"
                 IssueInstant="2023-09-18T22:04:33.800Z"
                 Version="2.0"
                 >...
    <saml2:Subject>
        ...
        <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
            <saml2:SubjectConfirmationData Address="               "
                                          NotOnOrAfter="2023-09-18T22:09:33.800Z"
                                          Recipient="https://auth.nih.gov/affwebser
                                          />
        </saml2:SubjectConfirmation>
    </saml2:Subject>
    <saml2:AuthnStatement AuthnInstant="2023-09-18T22:04:33.768Z"
                          SessionIndex="_316ea4d07a71cac98cdd438e210f4ab3"
                          >...
```

A user might be logged into the same web app on multiple devices, e.g., a team chat tool running on their computer and their phone. The SP must be able to link the user’s session on a particular device to the corresponding session at their IdP. Otherwise, logging out of the web app on one device might mistakenly log the user out of web apps on other devices.

To bind the right SP and IdP sessions together, the identity provider must include three pieces of information in every successful authentication

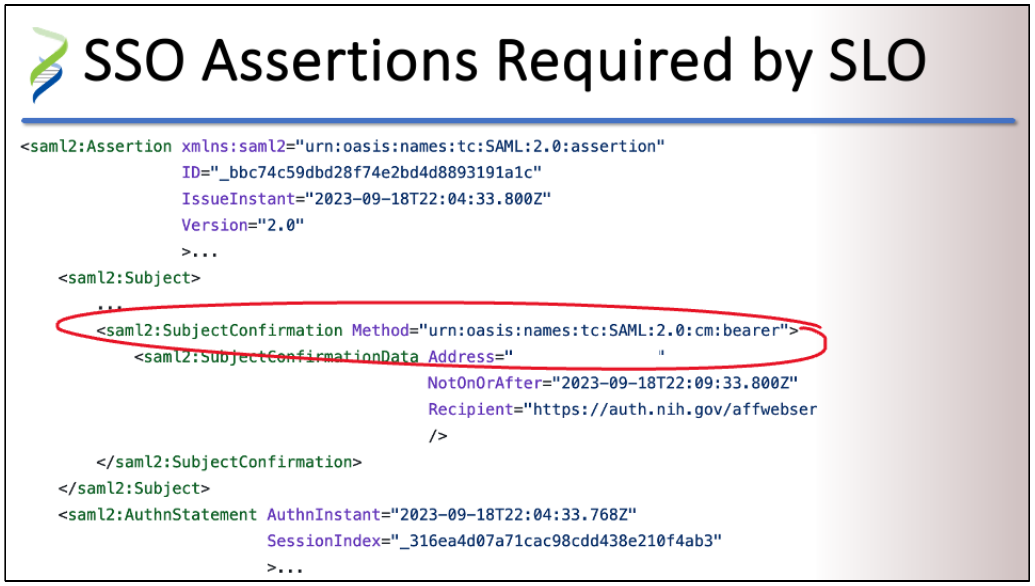
<Response>.



SSO Assertions Required by SLO

```
<saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
                 ID="_bbc74c59dbd28f74e2bd4d8893191a1c"
                 IssueInstant="2023-09-18T22:04:33.800Z"
                 Version="2.0"
                 >...
    <saml2:Subject>
        ...
        <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
            <saml2:SubjectConfirmationData Address="               "
                                           NotOnOrAfter="2023-09-18T22:09:33.800Z"
                                           Recipient="https://auth.nih.gov/affwebser
                                           />
        </saml2:SubjectConfirmation>
    </saml2:Subject>
    <saml2:AuthnStatement AuthnInstant="2023-09-18T22:04:33.768Z"
                          SessionIndex="_316ea4d07a71cac98cdd438e210f4ab3"
                          >...
```

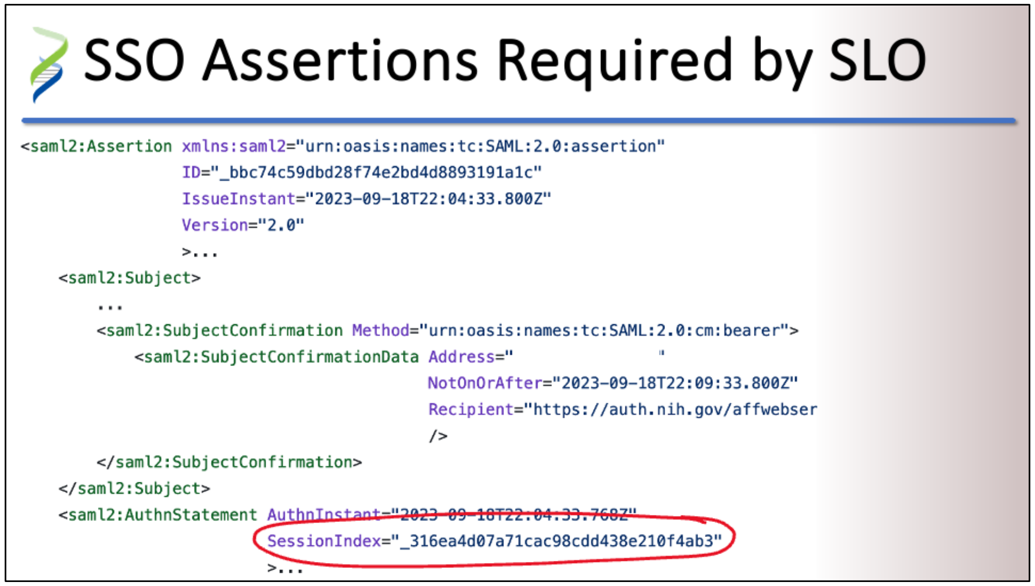
First, it must return an <Assertion> with a <Subject> and an <AuthnStatement> (authentication statement).



SSO Assertions Required by SLO

```
<saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
                 ID="_bbc74c59dbd28f74e2bd4d8893191a1c"
                 IssueInstant="2023-09-18T22:04:33.800Z"
                 Version="2.0"
                 >...
    <saml2:Subject>
        ...
        <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
            <saml2:SubjectConfirmationData Address="               "
                                           NotOnOrAfter="2023-09-18T22:09:33.800Z"
                                           Recipient="https://auth.nih.gov/affwebser
                                           />
        </saml2:SubjectConfirmation>
    </saml2:Subject>
    <saml2:AuthnStatement AuthnInstant="2023-09-18T22:04:33.768Z"
                          SessionIndex="_316ea4d07a71cac98cdd438e210f4ab3"
                          >...
```

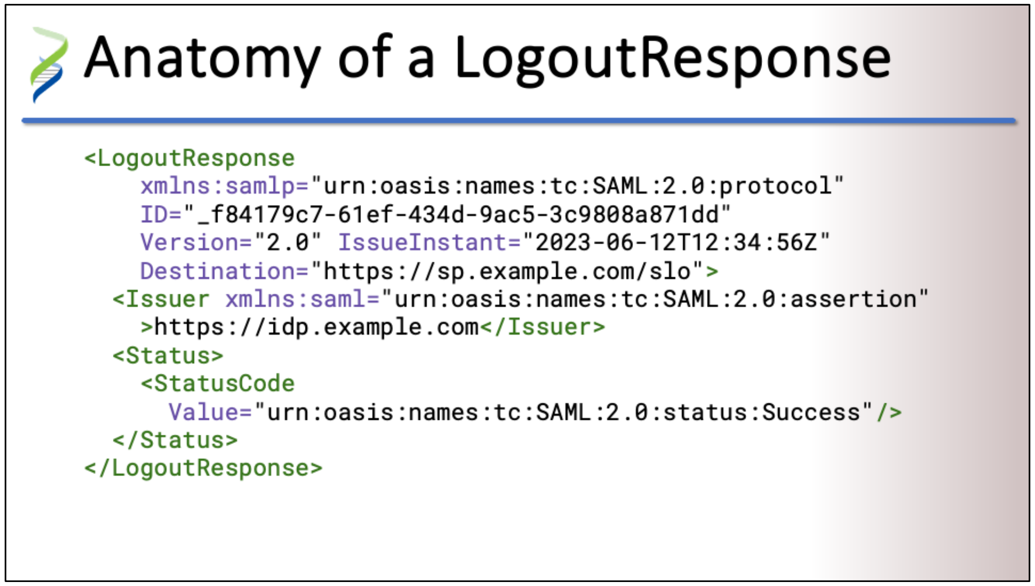
Second, the <Subject> must include a <SubjectConfirmation> with a Method set to “bearer”, which means that user themselves—not some third party—is involved in this session.



SSO Assertions Required by SLO

```
<saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
                 ID="_bbc74c59dbd28f74e2bd4d8893191a1c"
                 IssueInstant="2023-09-18T22:04:33.800Z"
                 Version="2.0"
                 >...
    <saml2:Subject>
        ...
        <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
            <saml2:SubjectConfirmationData Address="               "
                                           NotOnOrAfter="2023-09-18T22:09:33.800Z"
                                           Recipient="https://auth.nih.gov/affwebser
                                           />
        </saml2:SubjectConfirmation>
    </saml2:Subject>
    <saml2:AuthnStatement AuthnInstant="2023-09-18T22:04:33.760Z"
                          SessionIndex="_316ea4d07a71cac98cdd438e210f4ab3"
                          >...
```

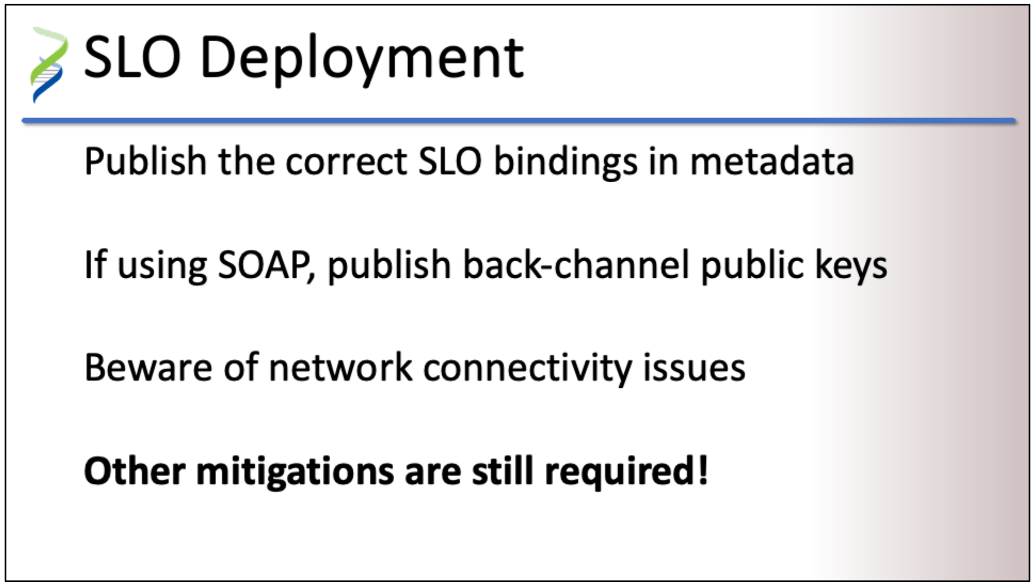
Third, the <AuthnStatement> must have a SessionIndex, which gives the service provider a handle on the user's session at the IdP. To protect the user's privacy, each SP will receive a unique SessionIndex. With this information, the SP can ask the IdP to log out the user on the same device they're using to access the web app.



Anatomy of a LogoutResponse

```
<LogoutResponse
    xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
    ID="_f84179c7-61ef-434d-9ac5-3c9808a871dd"
    Version="2.0" IssueInstant="2023-06-12T12:34:56Z"
    Destination="https://sp.example.com/slo">
  <Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
    >https://idp.example.com</Issuer>
  <Status>
    <StatusCode
      Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
  </Status>
</LogoutResponse>
```

A <LogoutResponse> message is structured like an authentication <Response>, but simpler, with a Destination, <Issuer>, and <StatusCode>.



SLO Deployment

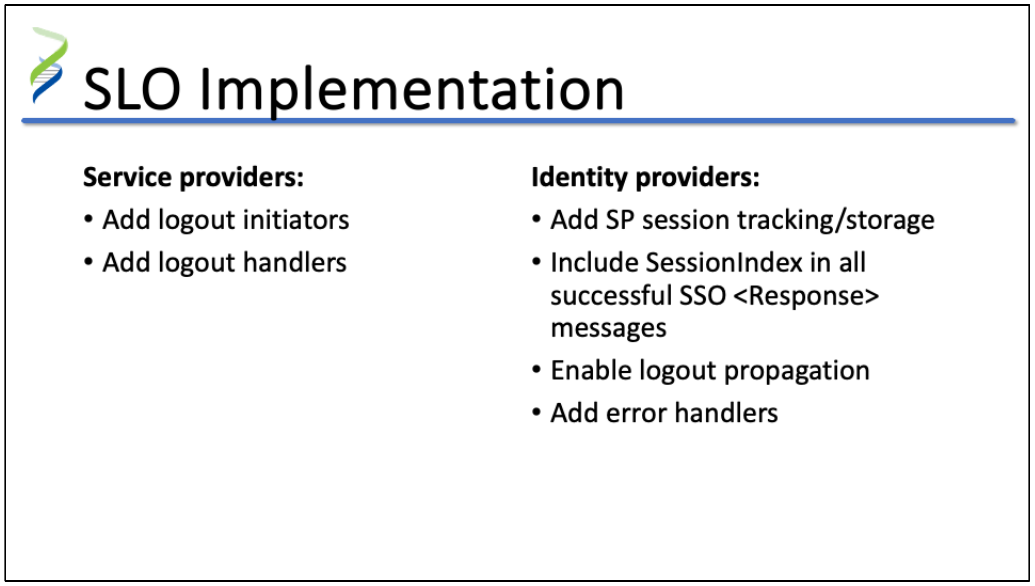
Publish the correct SLO bindings in metadata

If using SOAP, publish back-channel public keys

Beware of network connectivity issues

Other mitigations are still required!

Single Logout involves contacting multiple entities hosted in different environments by unrelated organizations. No one person or organization is in control. If one entity sends a <LogoutRequest> to another that doesn’t support SLO, the logout process will fail, leaving sessions active against the user’s expressed intent. Service providers, identity providers, and federation operators must make certain that federation metadata includes the correct SLO bindings to avoid this problem. If entities support back-channel presentation of logout requests, they must also publish the relevant public keys in federation metadata to facilitate SOAP endpoint authentication. Note, too, that network connectivity issues can interrupt the logout process in an unrecoverable manner. And even after all that, some entities simply will not support SLO at all, ever. This means Single Logout cannot replace other session hijacking mitigations such as user training to close the browser after a logout action or server-side automatic session termination.



SLO Implementation

Service providers:

- Add logout initiators
- Add logout handlers

Identity providers:

- Add SP session tracking/storage
- Include SessionIndex in all successful SSO <Response> messages
- Enable logout propagation
- Add error handlers

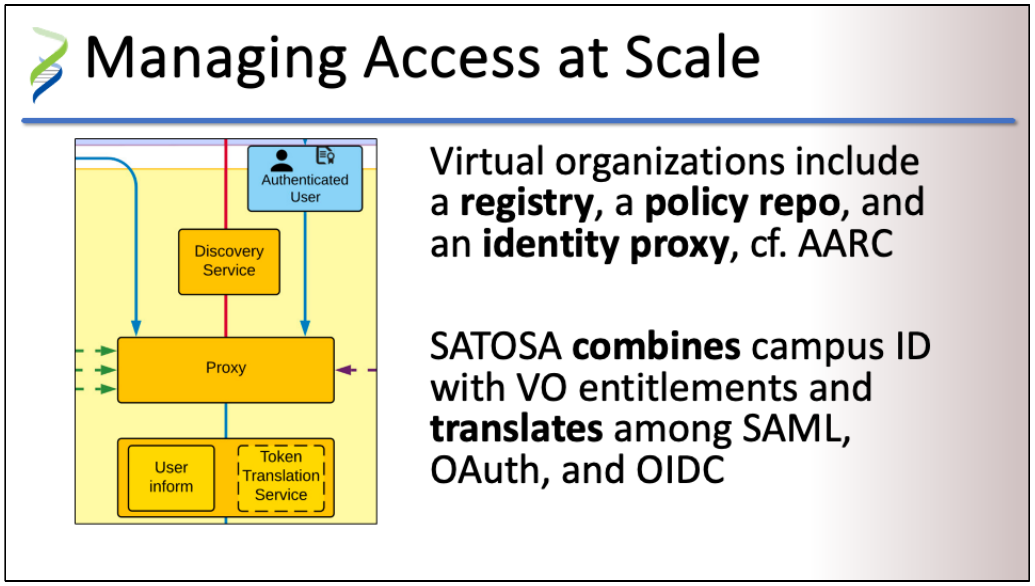
A service provider must be able to initiate a logout flow and handle any incoming logout requests or logout responses.

For an identity provider to implement Single Logout, it must keep track of which SPs the user accessed. The IdP must also include a SessionIndex in all SSO <Response> messages, which binds SP sessions to IdP sessions on the current device as mentioned. The IdP must also be capable of propagating logouts to SPs. This includes handling errors returned by those SPs during the logout flow and notifying users to take the appropriate actions.

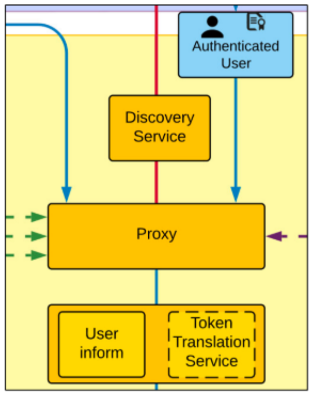
Note that automatic session termination should not trigger a logout flow.

RDCT

What’s SATOSA?



Managing Access at Scale



Virtual organizations include a **registry**, a **policy repo**, and an **identity proxy**, cf. AARC

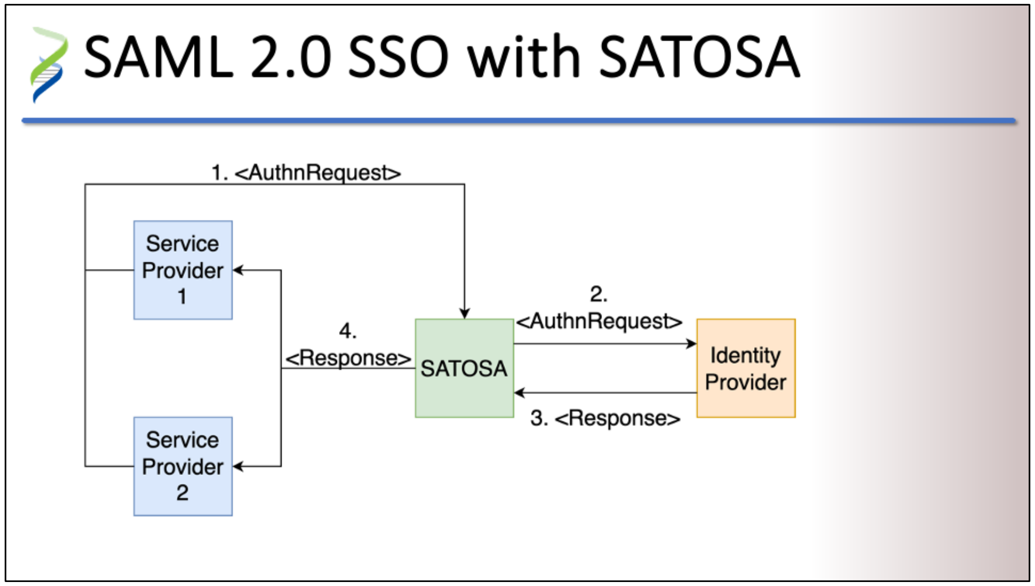
SATOSA **combines** campus ID with VO entitlements and **translates** among SAML, OAuth, and OIDC

Some research collaborations (such as LIGO or the NIAID Discovery Through Collaboration Platform) operate services for use across multiple institutions and cannot cede user identification or access management decisions to a single campus IdP. These collaborations need their own authentication and authorization infrastructure (AAI). The AARC Blueprint Architecture describes this in detail, but an AAI or virtual organization (VO) has three essential components:

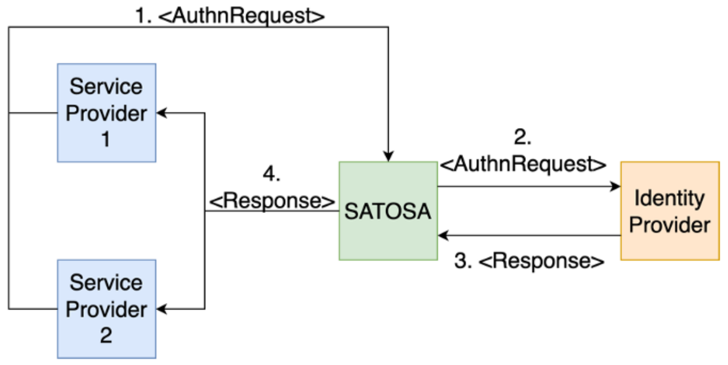
- a **user registry** that manages information about research collaboration members and their access rights
- a **policy repository** that controls access to end services based on users' entitlements

- an **identity proxy** that combines campus IdP-provided user information with the registry-managed rights assignments

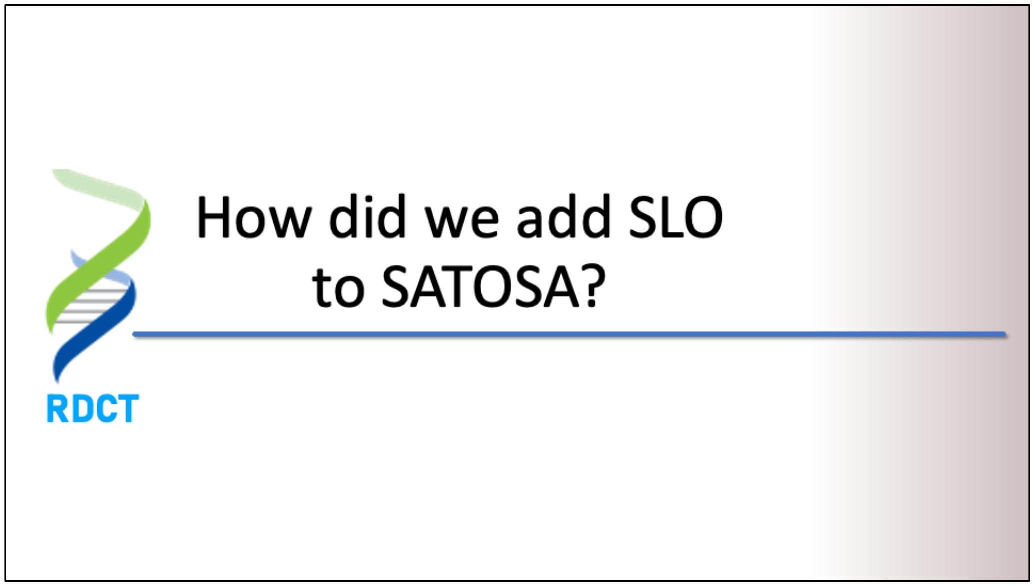
SATOSA is an identity proxy. Written in Python, it can translate authentication requests and responses among three different protocols—SAML, OAuth 2.0, and OpenID Connect. SATOSA implements SAML using the PySAML2 library.



SAML 2.0 SSO with SATOSA

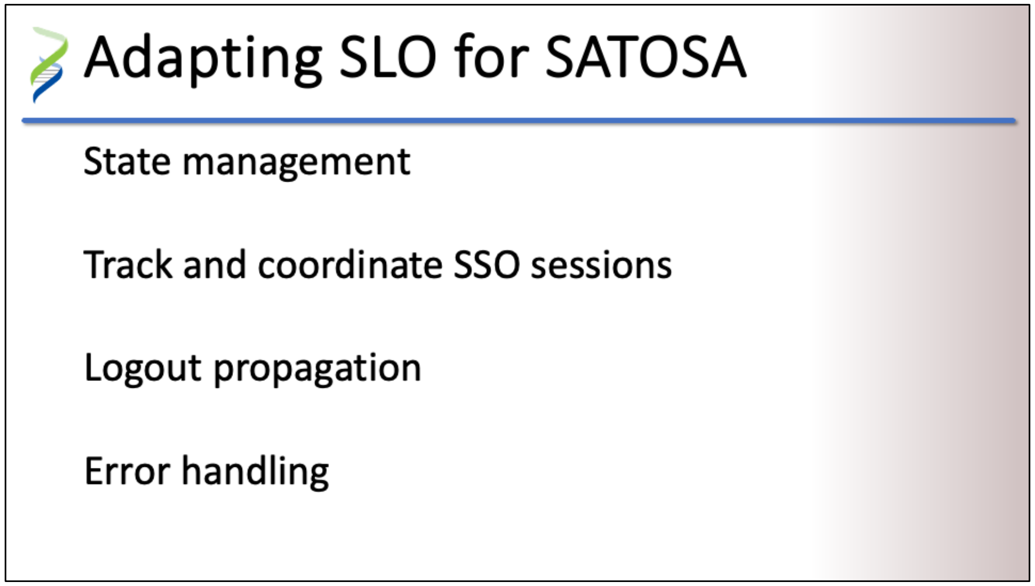


To the federation, SATOSA acts like a single service provider that represents the research collaboration. To the collaboration's web apps, it acts like a single identity provider representing the rest of the federation. SATOSA is stateless, which means it doesn't track proxied authentication requests or responses after they finish (successfully or not).



How did we add SLO to SATOSA?

RDCT



Adapting SLO for SATOSA

State management

Track and coordinate SSO sessions

Logout propagation

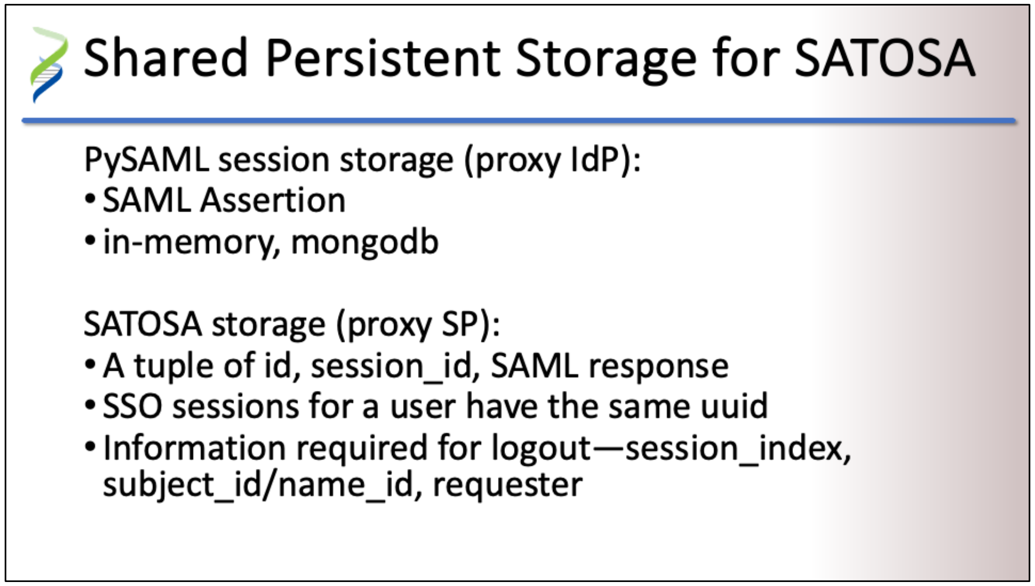
Error handling

To enable Single Logout profile to work for SATOSA, it is necessary to address the stateless nature of the proxy and implement changes to facilitate SLO.
Adaptations:

- Session management
  - Keep track of the sessions, expiry and timeout
  - Require/Extract the SessionIndex value from the SAML Response
  - Storage and deletion of the SAML Assertions
- Track and coordinate SSO sessions
  - Keep track of the users authentication status across accessed
- Add Single Logout endpoint handler to receive and process SLO messages
  - On the SAML frontend and backend
  - The proxy metadata should include SingleLogoutService endpoints
  - Provide support for Front-channel and Back-channel binding types
- Logout Propagation
  - Return more than one response for a single request

====
PySAML2 SLO support

- SP storage of SAML Assertion required for logout
- Supported SLO Binding Types in SATOSA



Shared Persistent Storage for SATOSA

PySAML session storage (proxy IdP):
- SAML Assertion
- in-memory, mongodb

SATOSA storage (proxy SP):
- A tuple of id, session_id, SAML response
- SSO sessions for a user have the same uuid
- Information required for logout—session_index, subject_id/name_id, requester

Since there are two IdPs, the proxy IdP and the campus IdP, two assertions are generated and need to be stored to construct the respective LogoutRequest messages. The campus IdP sends an Assertion to the proxy SP, and the proxy IdP sends an Assertion to each of the research collaboration's SPs. The proxy IdP is built with the PySAML2 server and has support for session storage, but the proxy SP is built with the PySAML2 client and currently has no support for session storage, hence the need to create a store for the Assertion info.

The ideal implementation would be to use/create SAML Backend storage and not create a data store in SATOSA itself.

The idea to use pysaml2 is borrowed from oidc implementation in satosa which is stateful.

To configure storage for SATOSA you must simply provide your database connection details and SATOSA will automatically take care of setting up the required tables.

Each authenticated request in SATOSA has:

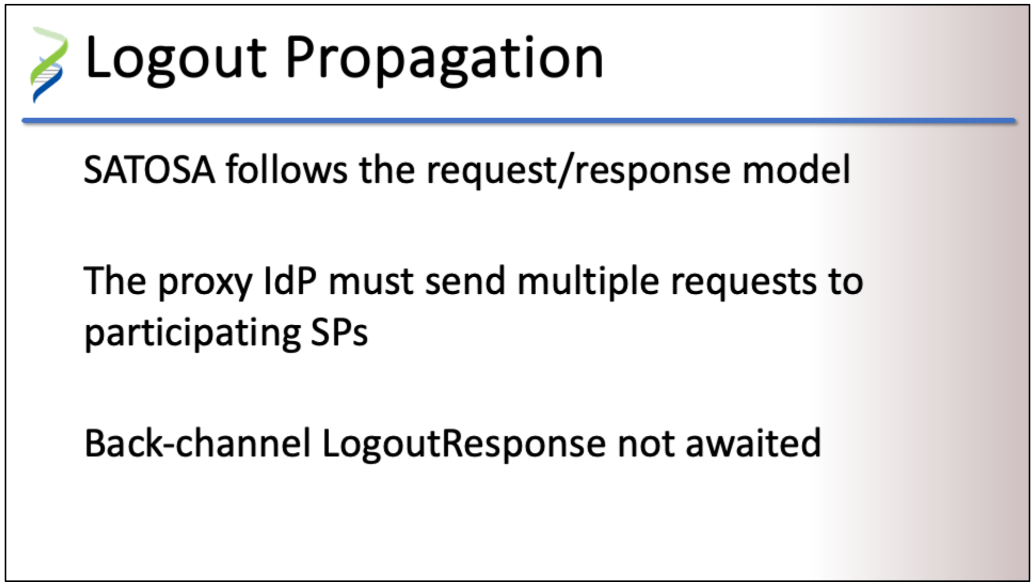
- unique identifier/primary key
- session_id (identify the user’s session)
- assertion information

authn state looks like:

```
{
  'auth_info': {
     'session_index':
['_7cb66bf7b89abfe29ab2f72e2ebfa256'],
     'auth_class_ref':
'urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtec
tedTransport',
     'timestamp': '2023-04-25T10:54:11.026Z',
     'issuer': 'https://samltest.id/saml/idp',
     'authority': None
  },
  'requester': 'https://example.org/shibboleth/sp',
  'requester_name': [{'text': None, 'lang': 'en'}],
  'subject_id':
```

```
'AAdzZWNyZXQx0uehqJgC/M3HJrpgeXMFm+havXxpEIRy
JQwBaEWN3K1laB707y2HKHEvF63jb8PA==',
  'subject_type': 'urn:oasis:names:tc:SAML:2.0:nameid-
format:transient',
  'attributes': {
'displayname': ['Sheldor'], 'givenname': ['Sheldon'],
'mail': ['scooper@samltest.id'], 'surname': ['Cooper'],
'uid': ['sheldon']
  }
}
```

(What happens to stored sessions for which the user has not initiated SLO?)



Logout Propagation

SATOSA follows the request/response model

The proxy IdP must send multiple requests to participating SPs

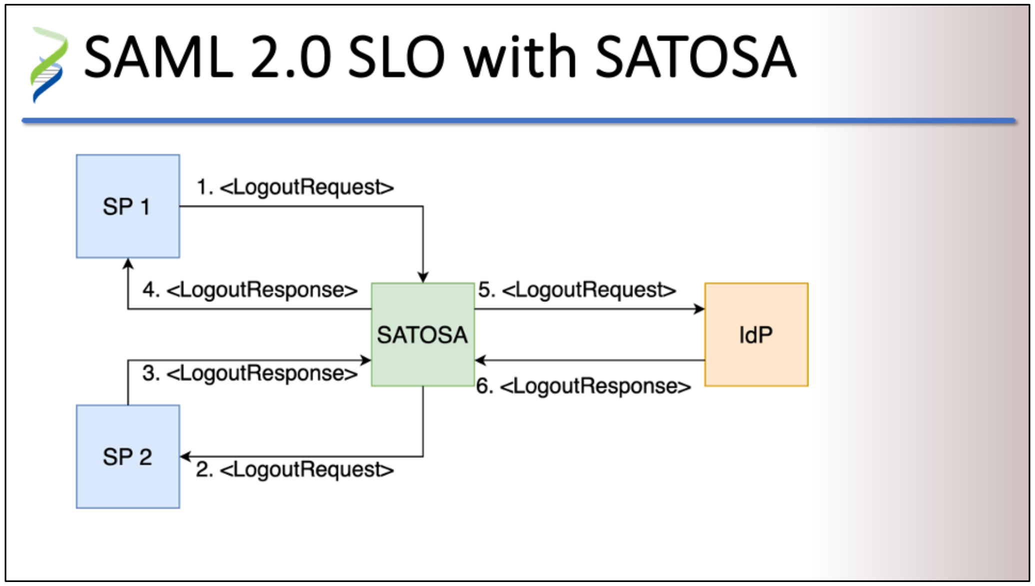
Back-channel LogoutResponse not awaited

SATOSA uses Gunicorn to receive inbound requests, route them to the appropriate handler, and return a response.

To make outbound requests as is required for logout propagation, we use the Python requests library.

When the proxy IdP receives a LogoutRequest, it generates a list of all SPs with an active SSO session and propagates LogoutRequest messages to each of them.

To do this, the store is queried using the session_id as a filter (session scope). For each SP in the list, SATOSA generates a <LogoutRequest>. It sends that message to the SP using its preferred binding type. To prevent stalling of the SLO process, SATOSA does not await LogoutResponses.



SAML 2.0 SLO with SATOSA

This diagram shows how Single Logout works in a SATOSA many-to-one configuration. Ideally, the SPs and IdP support SLO fully and have SLO endpoints in their metadata. The research collaboration's SPs are only aware of the session with the proxy IdP. Campus IdPs are only aware of the session with the proxy SP.

After SSO authentication, the campus IdP authentication Response sent to proxy SP is stored. SATOSA uses the session_id to identify the user's browser session. When the authentication response is routed to the proxy IdP, it stores the SSO assertion in the session storage.

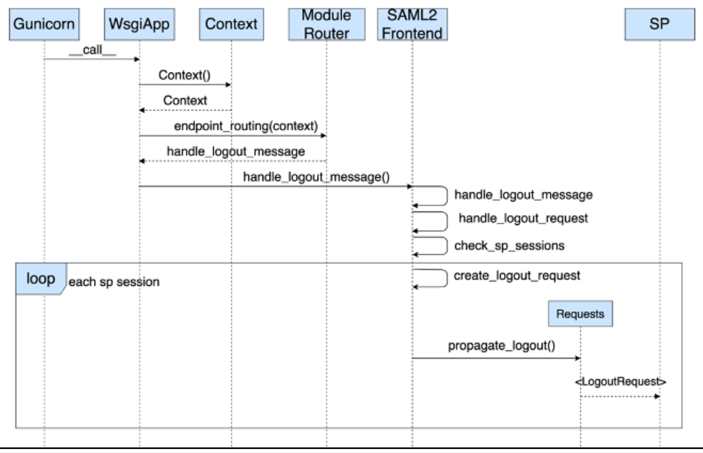
SP-initiated logout sends a LogoutRequest to the proxy

IdP, which checks the storage for other sessions using the same session_id. The proxy IdP then initiates logout for each SP with a session and deletes sessions from the database storage. When finished, the proxy IdP forwards the logout request to the proxy SP.

The proxy SP creates a LogoutRequest for the campus IdP. SATOSA selects a binding type and endpoint from those provided in the campus IdP's metadata. The IdP receives the LogoutRequest, processes it accordingly, and responds with a LogoutResponse.

When the proxy SP receives the LogoutResponse, it checks the StatusCode/Message. If the campus IdP returned a logout error, the proxy SP will return a SATOSA response message indicating an error to the user.

Proxy IdP LogoutRequest Handling



This simplified diagram showing how the proxy IdP handles LogoutRequest messages sent to its SingleLogoutService endpoints.

TODO: add callback

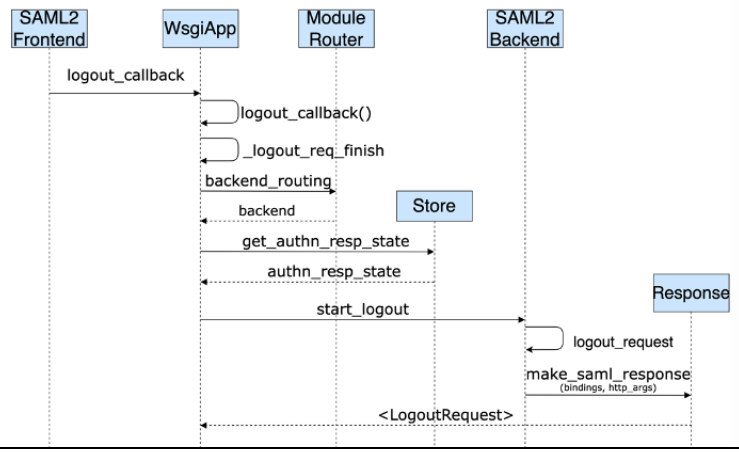
In SP-initiated Logout, the proxy IdP will receive the LogoutRequest. The proxy IdP must:

- Parse the LogoutRequest
- Look up the SP sessions that match the user’s session id in storage
- Send LogoutRequest messages to each SP matching the user
- Check whether the Initiating SP requires a response
    - Generate and return a LogoutResponse if required
- Delete the SP session from the database
- Call the logout callback function to redirect to the proxy SP

The proxy SP must:

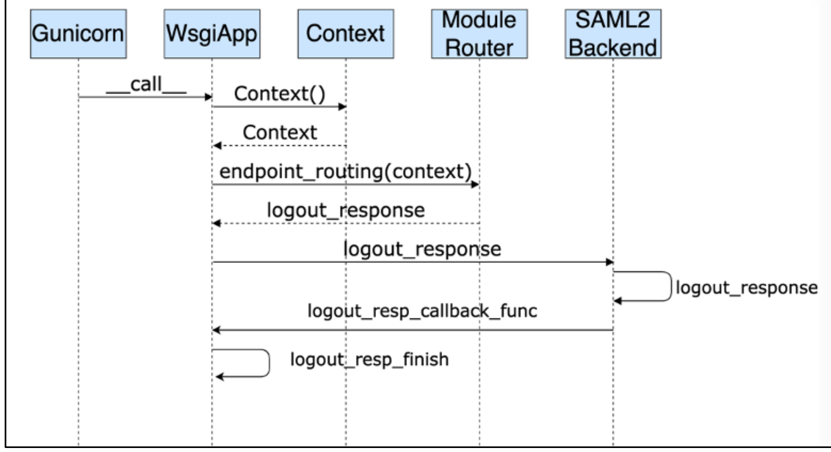
- Lookup the SP session information in the database
- Generate and send a LogoutRequest to the IdP

Proxy SP LogoutRequest Handling

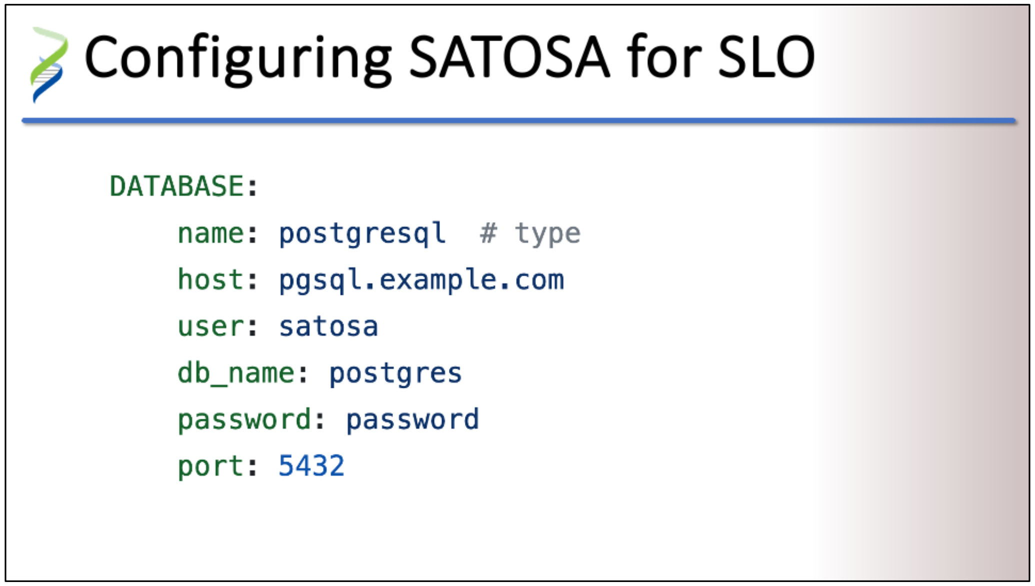


After logout is complete at the proxy IdP, the logout callback function will be called. The function will route the request to the proxy SP, which will initiate SLO at the IdP. The arguments for creating the LogoutResponse are retrieved from the SATOSA store and will include the NameID of the user and the SessionIndex that was sent in the original Assertion from the campus IdP. The proxy SP will then wait for a LogoutResponse from the campus IdP with details on the status of Logout. If the campus IdP is connected to other SPs, they may also receive LogoutRequests from the IdP if the user chooses to log out of all SPs.

Proxy SP LogoutResponse Handling



After the campus IdP terminates its SSO session, it will send a LogoutResponse to the proxy SP. Note that the browser session information is deleted, and SATOSA cannot rely on browser cookies to return information required to complete the state. When the proxy SP receives a LogoutResponse, the session_id is different the previous requests because the session was terminated. The logout_response handler will parse the LogoutResponse and check the StatusCode of the LogoutResponse. If it's a success status code, the proxy SP calls the Logoutback function. If it's an error, the proxy SP will return a HTTP response with error details.



Configuring SATOSA for SLO

```
DATABASE:
    name: postgresql  # type
    host: pgsql.example.com
    user: satosa
    db_name: postgres
    password: password
    port: 5432
```

SLO configuration is optional:

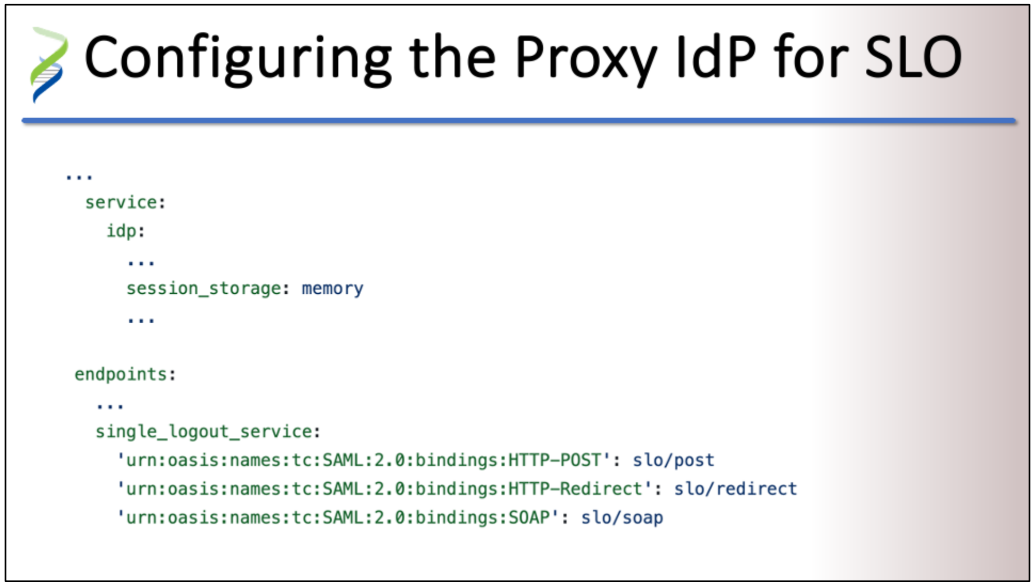
- Database is required
- SLO endpoints are optional
- pysaml2 server storage default is memory
- option to sign requests (but currently hard coded to true)

In the proxy_conf.yaml file, configure a database that will be used to store SLO state information.

Also:

- saml2_frontend
  - slo endpoints
  - session storage
- saml2_backend
  - slo endpoints
- microservices
  - custom frontend and backend that inherit from the base classes should add logout callback classes

Example proxy config:
https://gist.github.com/sebulibah/2e864689f891b43254373be575655633



Configuring the Proxy IdP for SLO

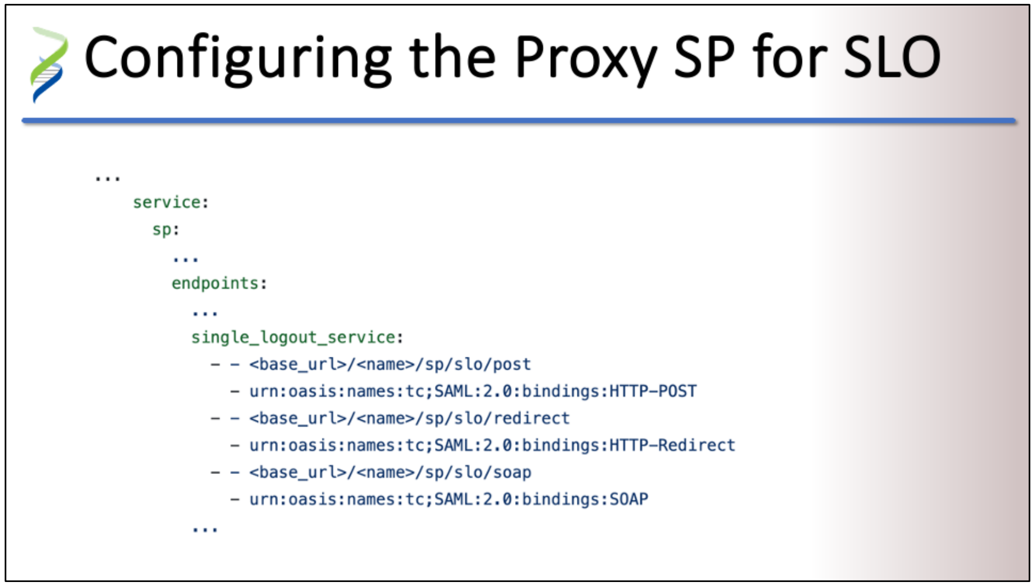
```
...
  service:
    idp:
      ...
      session_storage: memory
      ...

 endpoints:
   ...
   single_logout_service:
     'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST': slo/post
     'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect': slo/redirect
     'urn:oasis:names:tc:SAML:2.0:bindings:SOAP': slo/soap
```

To add the SLO configuration to the saml2 frontend you need to add the single logout service endpoints and configure session storage following the PySAML2 library guide. Both front channel and back channel bindings are supported.

The PySAML2 server can store assertions in memory or in MongoDB.

Example saml2_frontend config:
https://gist.github.com/sebulibah/ae628deceef06034b5e7c3001a801a17

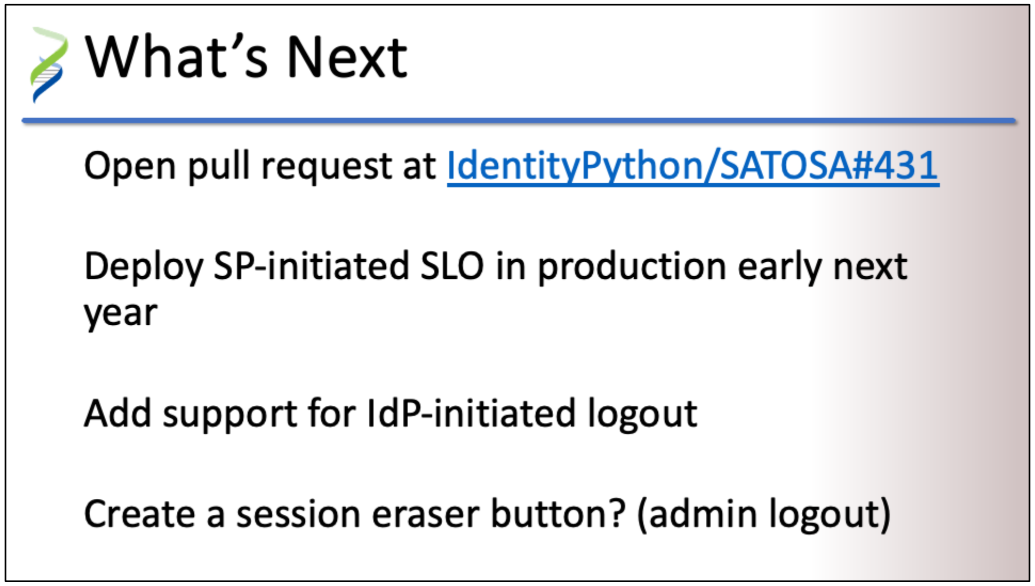


Configuring the Proxy SP for SLO

```
...
    service:
      sp:
        ...
        endpoints:
          ...
          single_logout_service:
            - - <base_url>/<name>/sp/slo/post
              - urn:oasis:names:tc;SAML:2.0:bindings:HTTP-POST
            - - <base_url>/<name>/sp/slo/redirect
              - urn:oasis:names:tc;SAML:2.0:bindings:HTTP-Redirect
            - - <base_url>/<name>/sp/slo/soap
              - urn:oasis:names:tc;SAML:2.0:bindings:SOAP
          ...
```

In the proxy SP, you will need to add single_logout_service endpoints. Again, both front channel and back channel bindings are supported. When running SATOSA for the first time, the metadata will be generated and will include the Single Logout Service endpoints.

Example saml2_backend config:
https://gist.github.com/sebulibah/41eb64788d8568d5da9efdef5729edf3



What's Next

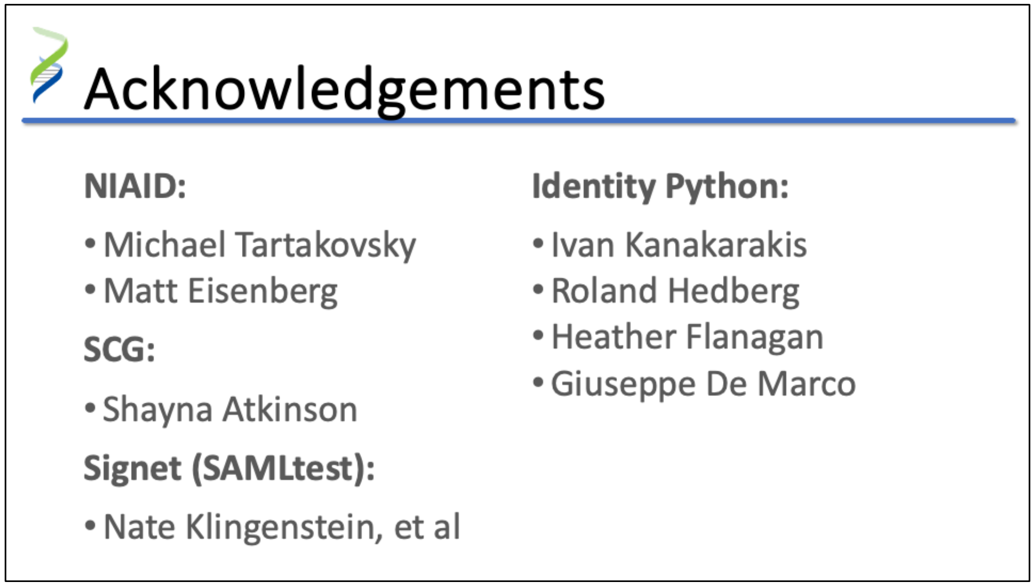
Open pull request at [IdentityPython/SATOSA#431]()

Deploy SP-initiated SLO in production early next year

Add support for IdP-initiated logout

Create a session eraser button? (admin logout)

TALK



Acknowledgements

NIAID:

- Michael Tartakovsky
- Matt Eisenberg

SCG:

- Shayna Atkinson

Signet (SAMLtest):

- Nate Klingenstein, et al

Identity Python:

- Ivan Kanakarakis
- Roland Hedberg
- Heather Flanagan
- Giuseppe De Marco

RDCT

Questions?