SaToSa Training
Training by AARC
Summary and Actions

✓ Training:
  ▪ What is SaToSa
  ▪ How to Install
  ▪ How to Configure
    ▪ directory
    ▪ Proxy_conf and internal_attributes
  ▪ Saml2
  ▪ OIDC
  ▪ Plugins & social
  ▪ How to extend (MS)
Summary and Actions

✓ Training:
  ▪ What is SaToSa
  ▪ How to Install
  ▪ How to Configure
    ▪ directory
    ▪ Proxy_conf and internal_attributes
  ▪ Saml2
  ▪ OIDC
  ▪ Plugins & social
  ▪ How to extend (MS)
  ▪ How to start
What is SaToSa

• A configurable proxy for translating between different authentication protocols
• Allows the manipulation of attributes and flows
• Based on Python3
• Easy to config
Many to one

- Many SP to a single IdP
One to many

• One
• Multiple Idp
  • Require a Discovery service
SAML2 to Social Login

- From SAML2 to Social Login
- One plugin for each social account
SaTosa allows translation between different protocols

- OpenID Connect <-> SAML2
- SAML2 <-> OpenID Connect

Later, we will see how to do that
What is SaToSa

Authentication protocols:

- SAML2
- OpenID Connect
- OAuth2
- Social Network (Facebook, Google, OrcID…)

Use Cases

- SAML2<->SAML2
- SAML2<->Social logins
- SAML2<->OIDC
- OIDC<->SAML2
Summary and Actions

✓ Training:
  ▪ What is SaToSa
  ▪ How to Install
  ▪ How to Configure
    ▪ directory
    ▪ Proxy_conf and internal_attributes
  ▪ Saml2
  ▪ OIDC
  ▪ Plugins & social
  ▪ How to extend (MS)
  ▪ How to start
How to install

- Two ways:
  - Docker
  - Manual installation
- Manual installation (First way)
  - i. Install dependencies: `apt-get install libffi-dev libssl-dev xmlsec1`
  - ii. Download the SATOSA proxy project as a compressed archive and unpack it to `<satosa_path>`.
  - iii. Install the application: "pip install `<satosa_path>`"
- Manual installation ("lazy" way)
  - "Pip install satosa"
- Docker is the recommended way of running the proxy
  - LINK: https://hub.docker.com/r/satosa/satosa/
Docker command

Docker pull command:
Docker pull satosa/satosa
docker run
  -p <port on host>:<proxy_port>
  -v <host directory>:<data_dir>
  -e DATA_DIR=<data_dir>
  -e PROXY_PORT=<proxy_port>
  [-e METADATA_DIR=<metadata_dir>]
satosa/satosa
Summary and Actions

✓ Training:
  ▪ What is SaToSa
  ▪ How to Install
  ▪ How to Configure
    ▪ Directory
    ▪ Proxy_conf and internal_attributes
    ▪ SamI2
    ▪ OIDC
    ▪ Plugins & social
  ▪ How to extend (MS)
  ▪ How to start
What is SaToSa / Example directory

- Proxy.conf
- Internal_attributes.yaml
- plugins/
  - Backends/
    - Saml2_backends.yaml
    - Google_backends.yaml
    - Facebook_backends.yaml
    - ..._backends.yaml
  - Frontends/
    - Openid_connect_frontend.yaml
    - saml2_frontend.yaml
  - Microservices/
    - Account_linking.yaml
    - ldap_attributes.yaml
File hierarchy
Proxy_conf

- Configuration file. It points to all satosa files and modules
- Provide list of directory/file path, to enable any module
  - Frontend
  - Backend
  - Microservices
  - Plugins

<table>
<thead>
<tr>
<th>BASE</th>
<th>base url of the proxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>COOKIE_STATE_NAME</td>
<td>name of cookie SATOSA uses for preserving state between requests</td>
</tr>
<tr>
<td>STATE_ENCRYPTION_KEY</td>
<td>key used for encrypting the state cookie, will be overridden by the environment variable SATOSA_STATE_ENCRYPTION_KEY if it is set</td>
</tr>
<tr>
<td>INTERNAL_ATTRIBUTES</td>
<td>path to attribute mapping</td>
</tr>
<tr>
<td>CUSTOM_PLUGIN_MODULE_PATHS</td>
<td>list of directory paths containing any front-/backend plugin modules</td>
</tr>
<tr>
<td>BACKEND_MODULES</td>
<td>list of plugin configuration file paths, describing enabled backends</td>
</tr>
<tr>
<td>FRONTEND_MODULES</td>
<td>list of plugin configuration file paths, describing enabled frontends</td>
</tr>
<tr>
<td>MICRO_SERVICES</td>
<td>list of plugin configuration file paths, describing enabled microservices</td>
</tr>
<tr>
<td>USER_ID_HASH_SALT</td>
<td>salt used when creating the persistent user identifier, will be overridden by the environment variable SATOSA_USER_ID_HASH_SALT if it is set</td>
</tr>
<tr>
<td>LOGGING</td>
<td>optional configuration of application logging</td>
</tr>
</tbody>
</table>
Internal Attributes

- Map every internal attributes
- Every internal attribute has a map of profiles, which in turn has a list of external attributes names which should be mapped to the internal attributes
- multiple external attributes are specified under a profile
- “User_id_from_attrs” override user identifier generated by the backend module with a list of internal attribute names
- “User_id_to_attr” store the user identifier in a specific internal attribute

```
attributes:
  mail:
    openid: [email]
    saml: [mail, emailAdress, email]
  address:
    openid: [address.formatted]
    saml: [postaladdress]
```
Plugins

• Divided into:
  • frontends, receiving requests from clients
  • backends, sending requests to target providers
  • Micro_services, allows the management and manipulation of attributes
• Require usually 3 parameters:
  • Module, module file path
  • Name, unique name to identify this plugin
  • Config, provide variable to make plugin work correctly
• plugins are customizable
SAML2 Plugin

- SAML2 frontend acts as a SAML Identity Provider (IdP)
  - SAML2 backend acts as a SAML Service Provider (SP), making authentication requests to SAML Identity Providers (IdP)
- The SAML2 frontend comes in 3 different flows:
  - “SAMLMirrorFrontend” module, mirrors each target provider as a separate entity in the SAML metadata
    SP -> optional discovery service -> selected proxy SAML entity -> target IdP
  - “SAMLFrontend” module, acts like a single IdP, and hides all target providers
    SP -> proxy SAML SSO location -> target IdP
- SAML frontend can also further restrict the attribute release
<table>
<thead>
<tr>
<th><strong>Saml2 Plugin\Parameters</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>organization</strong></td>
</tr>
<tr>
<td><strong>contact_person</strong></td>
</tr>
<tr>
<td><strong>key_file</strong></td>
</tr>
<tr>
<td><strong>cert_file</strong></td>
</tr>
<tr>
<td><strong>metadata[&quot;local&quot;]</strong></td>
</tr>
<tr>
<td><strong>attribute_profile</strong></td>
</tr>
<tr>
<td><strong>entityid_endpoint</strong></td>
</tr>
<tr>
<td><strong>acr_mapping</strong></td>
</tr>
</tbody>
</table>

- **organization**
  - display_name: Example Identities
  - name: Example Identities Organization
  - url: https://www.example.com

- **contact_person**
  - contact_type: technical
  - given_name: Someone Technical
  - email_address: technical@example.com

- **key_file**: path to private key used for signing (backend)/decrypting (frontend) SAML2 assertions
- **cert_file**: path to certificate for the public key associated with the private key in `key_file`
- **metadata["local"]**: list of paths to metadata for all service providers (frontend)/identity providers (backend) communicating with the proxy
- **attribute_profile**: attribute profile to use for mapping attributes from/to response
- **entityid_endpoint**: whether `entityid` should be used as a URL that serves the metadata XML document
- **acr_mapping**: custom Authentication Context Class Reference
Saml2 Frontend - Backend Plugin\Metadata

Metadata from local file:

"metadata":
    local: [idp.xml]

Metadata from remote URL:

"metadata": {
    "remote":
        https://example.org/simplesaml/module.php/aggregator/ : null
}

Metadata from remote mdq:

"metadata": {
    "mdq":
        https://example.disco.org: null
}
module: satosa.frontends.saml2.SAMLFrontend
name: Saml2IDP
config:
  idp_config:
    organization: {display_name: Example Identities, name: Example Identities Org., url: 'http://www.example.com'}
    contact_person:
      - {contact_type: technical, email_address: technical@example.com, given_name: Technical}
      - {contact_type: support, email_address: support@example.com, given_name: Support}
  key_file: frontend.key
  cert_file: frontend.crt
  metadata:
    local: [sp.xml]

entityid: <base_url>/<name>/proxy.xml
accepted_time_diff: 60
service:
  idp:
    endpoints:
      single_sign_on_service: []
    name: Proxy IdP
  ui_info:
    display_name:
      - lang: en
        text: "IdP Display Name"
    description:
      - lang: en
        text: "IdP Description"
    information_url:
      - lang: en
        text: "http://idp.information.url/"
    privacy_statement_url:
      - lang: en
        text: "http://idp.privacy.url/"
  keywords:
    - lang: se
      text: ["Satos", "IdP-SE"]
    - lang: en
      text: ["Satos", "IdP-EN"]
  logo:
    text: "http://idp.logo.url/"
    width: "100"
    height: "100"
  name_id_format: [urn:oasis:names:tc:SAML:2.0:nameid-format:person]
  policy:
    default:
      attribute_restrictions: null
      fail_on_missing_requested: false
      lifetime: {minutes: 15}
      name_form: urn:oasis:names:tc:SAML:2.0:attrname-format:uri
endpoints:

single_sign_on_service: {
  'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST': sso/post,
  'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect': sso/redirect
}
module: satosa.backends.saml2.SAMLBackend
name: SAML
config:
  idp_blacklist_file: /path/to/blacklist.json
  sp_config:
    key_file: backend.key
    cert_file: backend.crt
  organization: {display_name: Example Identities, name: Example Identities Org., url: 'http://www.example.com'}
  contact_person:
    - {contact_type: technical, email_address: technical@example.com, given_name: Technical}
    - {contact_type: support, email_address: support@example.com, given_name: Support}
metadata:
  local: [idp.xml]
entityid: <base_url>/name/proxy_saml2_backend.xml
accepted_time_diff: 60
service:
  sp:
    ui_info:
      display_name:
        - lang: en
          text: "SP Display Name"
      description:
        - lang: en
          text: "SP Description"
      information_url:
        - lang: en
          text: "http://sp.information.url/"
      privacy_statement_url:
        - lang: en
          text: "http://sp.privacy.url/"
      keywords:
        - lang: se
          text: ["Satos", "SP-SE"]
        - lang: en
          text: ["Satos", "SP-EN"]
    logo:
      text: "http://sp.logo.url/
      width: "100"
      height: "100"
    want_response_signed: true
    allow_unsolicited: true
    endpoints:
      assertion_consumer_service:
        - [base_url]/<name>/acs/post, 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST'
        - [base_url]/<name>/acs/redirect, 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect'
    discovery_service:
      name_id_format: 'urn:oasis:names:tc:SAML:2.0:nameid-format:transient'
# disco_srv must be defined if there is more than one IdP in the metadata specified above
disco_srv: http://disco.example.com
OIDC Plugin

• OpenID Connect backend acts as an OpenID Connect Relying Party (RP), making authentication requests to OpenID Connect Provider (OP)
  • supports discovery and dynamic client registration
• OpenID Connect frontend acts as and OpenID Connect Provider (OP), accepting requests from OpenID Connect Relying Parties (RPs).
  • this plugin is NOT stateless
module: satosa.frontends.openid_connect.OpenIDConnectFrontend
name: OIDC
config:
  signing_key_path: frontend.key
db_uri: mongodb://db.example.com # optional: only support MongoDB, will default to in-memory storage if not specified
client_db_path: /path/to/your/cdb.json
provider:
  client_registration_supported: Yes
  response_types_supported: ["code", "id_token token"]
  subject_types_supported: ["pairwise"]
  scopes_supported: ["openid", "email"]
module: satosa.backends.openid_connect.OpenIDConnectBackend
name: openid_connect
config:
  provider_metadata:
    issuer: https://op.example.com
  client:
    auth_req_params:
      response_type: code
      scope: [openid, profile, email, address, phone]
  client_metadata:
    application_name: SATOSA
    application_type: web
  contacts: [ops@example.com]
  redirect_uris: [<base_url>/]<name>
  subject_type: public

description:
entity_info:
  contact_person:
    - contact_type: "technical"
      email_address: ["technical_test@example.com", "support_test@example.com"]
    given_name: "Test"
    sur_name: "OP"
  - contact_type: "support"
    email_address: ["support_test@example.com"]
    given_name: "Support Test"
  organization:
    display_name:
      - ["OP Identities", "en"]
    name:
      - ["En test-OP", "se"]
      - ["A test OP", "en"]
    url:
      - ["http://www.example.com", "en"]
      - ["http://www.example.se", "se"]
  ui_info:
    description:
      - ["This is a test OP", "en"]
    display_name:
      - ["OP - TEST", "en"]
Social login

- Social login plugins can be used as backends for the proxy, allowing the proxy to act as a client to the social login services.
- Available social:
  - Google
  - Facebook
  - Github
  - Linkedin
  - OrcID
  - Oauth
Training:
- What is SaToSa
- Who is SaToSa
- How to Install
- How to Configure
  - directory
  - Proxy_conf and internal_attributes
  - Saml2
  - OIDC
  - Plugins & social
- How to extend (MS)
- How to start
Micro services

- Micro services allow additional behaviour, configured inside proxy.
- Two different types of micro services:
  - request micro services, which are applied to the incoming request
  - response micro services, which are applied to the incoming response from the target provider.
- Bundled micro services in SaToSa:
  - AddStaticAttributes
  - FilterAttributeValues
  - DecideBackendByRequester
  - DecideIfRequesterIsAllowed
  - Account linking
  - User consent management
  - LDAP attribute store
Custom plugins

- It's possible to write custom plugins which can be loaded by SaToSa
- Depending on which type of plugin it is, it has to inherit from the correct base class and implement the specified methods:
  - Frontends must inherit satosa.frontends.base.FrontendModule
  - Backends must inherit satosa.backends.base.BackendModule
  - Request micro services must inherit satosa.micro_services.base.RequestMicroService
  - Response micro services must inherit satosa.micro_services.base.ResponseMicroService
Summary and Actions

✔ Training:
  - What is SaToSa
  - Who is SaToSa
  - How to Install
  - How to Configure
    - directory
    - Proxy_conf and internal_attributes
    - Saml2
    - OIDC
    - Plugins & social
  - How to extend (MS)
  - How to start
Generate metadata

- Proxy metadata is generated based on the front-/backend plugins listed in `proxy_conf.yaml` using the `satosa-saml-metadata`
  - installed globally by SATOSA installation
- `satosa-saml-metadata path to proxy_conf.yaml path to key for signing path to cert for signing`
Running proxy application

• SATOSA proxy is a Python WSGI application and so it requires to be run using any WSGI compliant web server.
• Different solutions:
  • Using Gunicorn
  • Using Apache HTTP Server and mod_wsgi
Gunicorn

- Python WSGI HTTP Server for UNIX
- Often proxied by a full featured general purpose web server (Nginx or Apache) for:
  - to help buffer slow clients
  - To enable more sophisticated error page rendering
  - To handle SSL sessions
- Start with the following command:
  - `gunicorn -b<socket address> satosa.wsgi:app --keyfile=<https key> --certfile=<https cert>`
Apache HTTP Server and mod_wsgi

- Full guide available at the following link:
Thank you

Any Questions?

https://aarc-project.eu

© GEANT on behalf of the AARC project

The work leading to these results has received funding from the European Union’s Horizon 2020 research and innovation programme under Grant Agreement No. 730941 (AARC2).