Authentication

- Protocols (CAS, SAML)
  - CAS
    - CAS Definitions
    - CAS Server
    - CAS Clients
    - Authentication Flow
    - Validation
    - Attribute release
    - UGA implementation
  - SAML
    - Elements
    - Metadata
    - Authentication Flow
    - UGA implementation
  - Multifactor Authentication (Archpass)
  - Sample Configuration
  - Testing the Configuration
  - UGA Application Listserv
  - UGA SSO Maintenance Windows
    - Development Environments
    - Stage Environments
    - Production Environments

Enterprise Authentication

EITS provides multiple services to accommodate central authentication for UGA and its partners to authenticate using MyID credentials and, optionally, ArchPass two factor authentication powered by Duo. Each service is comprised of an authentication server and an application client which redirects calls for authentication to the authentication server. The client is installed and supported by the application administrators. EITS Identity management can offer only limited assistance for the client as each application may differ in the client implementation.

Please bookmark this page for service troubleshooting, also see our FAQ.

Protocols (CAS, SAML)

**CAS**

The Central Authentication Service (CAS) is a single sign-on protocol for the web which permits a user to access multiple applications while providing their credentials only once. It also allows web applications to authenticate users without gaining access to a user's security credentials, such as a password. The CAS service is UGA's primary authentication interface.

**CAS Definitions**

- Client: refers to the end user adj/or the web browser
- CAS Client: refers to the software component that is integrated with a web application and interacts with a CAS server via the CAS protocol
- Server: refers to the CAS server
- Service: refers to the application the client is trying to access
- The TGT (Ticket Granting Ticket), stored in the CASTGC cookie, represents a SSO session for a specific user.
- The ST (Service Ticket), transmitted as a GET parameter in URLs, stands for the access granted by the CAS server to the CASified application for a specific user.

**CAS Server**

- The CAS server is a Java servlet whose primary responsibility is to authenticate users and grant access to CAS-enabled services, commonly called CAS clients, by issuing and validating tickets. An Single Sign On (SSO) session is created when the server issues a ticket to the user upon successful login. A service ticket (ST) is issued to a service at the user's request via browser. The ST is subsequently validated at the CAS server via back-channel communication.

**CAS Clients**

- A CAS client is a software package that can be integrated with various software platforms and applications in order to communicate with the CAS server via some authentication protocol (e.g. CAS, SAML, OAuth).
- Platforms:
  - Apache httpd Server (mod_auth_cas module)
  - Java (Java CAS Client)
  - .NET (.NET CAS Client)
  - PHP (phpCAS)
  - Perl (PerlCAS)
• Python (pycas)
• Ruby (rubycas-client)

Applications:
• Outlook Web Application (ClearPass + .NET CAS Client)
• Atlassian Confluence
• Atlassian JIRA
• Drupal
• Liferay
• uPortal

Clients developed for various applications and platforms can be downloaded here: https://apereo.github.io/cas/6.0.x/planning/Architecture.html#cas-clients

The relationship between the CAS client, the CAS server and the user's browser can be seen in the following diagram:

The relationship between the application server, web browser and CAS server is depicted in the following diagram:

Authentication Flow
When the Application server's CAS client initiates authentication, it begins by redirecting the web browser to the CAS server's login service with one or more parameters: https://cas.example.org/cas/login?service=http%3A%2F%2Fwww.example.org%2Fservice. There are parameters within this URL that can be specified—required or optional. Some of the more common ones are as follows: The login service generates a service ticket.

- gateway [OPTIONAL] - if this parameter is set to true then the user will not be asked to re-enter their credentials. An important observation is that the 'gateway' and 'renew' parameters are not to be used together. In the event they are, behavior is undefined.
- service - though technically optional, this is the identifier of the application the browser is trying to access. In almost all cases, this will be the URL of the application. As an HTTP request parameter, this URL value MUST be URL-encoded as described in section 2.2 of RFC 3986. If a service is not specified, a service ticket cannot be issued and application sign on will not be able to occur.
  - For instance if we wanted to reach a website called "http://www.example.org", that would be our service and our URL would look like: https://cas.example.org/cas/login?service=http%3A%2F%2Fwww.example.org
- renew [OPTIONAL] - if this parameter is set to true, single sign-on will be bypassed. In this case, CAS will require the client to present credentials regardless of the existence of a single sign-on session with CAS and the user will have to log in each time.
- method [OPTIONAL, CAS 3.0] - The method to be used when sending responses. While native HTTP redirects (GET) may be utilized as the default method, applications that require a POST response can use this parameter to indicate the method type.

**Validation**

- Standard CAS service ticket validation occurs when the application server makes a request to the CAS server's /serviceValidate service with multiple parameters. The following discussions only /serviceValidate [CAS 2.0]:

(Please refer here for more information on /validate [CAS 1.0])

/serviceValidate checks the validity of a service ticket and returns an XML-fragment response. /serviceValidate MUST also generate and issue proxy-granting tickets when requested. /serviceValidate MUST NOT return a successful authentication if it receives a proxy ticket. It is RECOMMENDED that if /serviceValidate receives a proxy ticket, the error message in the XML response SHOULD explain that validation failed because a proxy ticket was passed to /serviceValidate. /serviceValidate can take multiple parameters that can be required or optional. Some of the more common ones are as follows:

- service [REQUIRED] - identifies the service for which the ticket was issued. As an HTTP parameter, this value must be URL encoded.
- ticket [REQUIRED] - the service ticket generated by the login service
  - https://cas.example.org/cas/serviceValidate?service=http%3A%2F%2Fwww.example.org%2Fservice&ticket=ST-1856339-aA5Yuvrxzp8T8u1cYG7
- renew [OPTIONAL] - if this parameter is set, ticket validation will only succeed if the service ticket was issued from the presentation of the user's primary credentials. It will fail if the ticket was issued from a single sign-on session.
  - https://cas.example.org/cas/serviceValidate?service=http%3A%2F%2Fwww.example.org%2Fservice&ticket=ST-1856339-aA5Yuvrxzp8T8u1cYG7&renew=true
- In the even that renew=true, if the service ticket was not generated by 'www.example.org', validation will fail.
- pgtUrl [OPTIONAL] - the URL of the proxy callback

**Attribute release**

The CAS service supports the release of a limited number of identity attributes via the SAML 1.1 and 2.0 protocols. Application owners will need to work with the application support vendors to properly map the attributes and ensure the proper access get provisioned. The following attributes are currently available to release via SAML 1.1 and 2.0 through our enterprise CAS service. These are the labels as the attributes are released:

- CN (Common name, should be same as MyID)
- DN (Fully qualified distinguished name from MSMyID )
- firstName
- lastName
- If you require more information, please elaborate on the access form.

NOTE: The CAS service provides support for the SAML 2.0 protocol with backwards compatibility for SAML 1.1. New applications will default to CAS 2.0.

In order to retrieve released SAML attributes, the application server must validate the service ticket by posting a request to the /samlValidate service. The following case-sensitive parameter must be included in the request to /samlValidate:

- Target [REQUIRED] - URL encoded service identifier of the back-end service. Note that as an HTTP request parameter, this URL value MUST be URL-encoded. The service identifier specified here MUST match the service parameter provided to /login. The TARGET service SHALL use HTTPS. SAML attributes MUST NOT be released to a non-SSL site.
- The request to /samlValidate MUST be an HTTP POST request. The request body MUST be a valid SAML 1.0 or 1.1 request XML document of document type "text/xml". The following SAML request values must be included in the SAML request body:
  - RequestID [REQUIRED] - unique identifier for the request
  - IssuerInstant [REQUIRED] - timestamp of the request
  - samlp:AssertionArtifact [REQUIRED] - the valid CAS Service Ticket obtained as a response parameter at login.
The following is an example of a valid POST request to /samlValidate:

Header

```plaintext
POST /cas/samlValidate?TARGET=
Host: cas.example.com
Content-Length: 491
Content-Type: text/xml
```

SAML 1.1 Request Body

```xml
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
      <samlp:AssertionArtifact>ST-1-u4hrm3td92cLxpCvrjylcas.example.com</samlp:AssertionArtifact>
    </samlp:Request>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

UGA implementation

The UGA enterprise CAS service currently utilizes Jasig CAS Version 6.0. We maintain 3 instances for development, staging, and production. These instances are at the following addresses:

- **Development** – [sso.dev.uga.edu](https://uga.teamdynamix.com/TDClient/Requests/ServiceDet?ID=30597) (uses dev credentials and may not be feasible for most testing. More information can be found here for test accounts: [https://uga.teamdynamix.com/TDClient/Requests/ServiceDet?ID=30597](https://uga.teamdynamix.com/TDClient/Requests/ServiceDet?ID=30597))
- **Stage** – sso.stage.uga.edu
- **Production** – sso.uga.edu

Registration of Service IDs for CAS authorization can be requested with the following form: [SSO Integration Request](https://uga.teamdynamix.com/TDClient/Requests/ServiceDet?ID=30597). Newly registered applications will be subject to review and vulnerability scanning by the EITS Information Security department.

**SAML**

Security Assertion Markup Language (SAML, pronounced sam-el) is an open standard for exchanging authentication and authorization data between parties, in particular, between an identity provider and a service provider. As its name implies, SAML is an XML-based markup language for security assertions.

**Elements**

- **Web Browser** – represents the user within the SSO process. The SAML protocol relies on HTTP headers and HTTP redirects.
- **Identity Provider (IdP)** – Sometimes called an Identity Service Provider or Identity Assertion Provider, it is an online service or website that authenticates users on the Internet by means of security tokens. Service Providers depend on an Identity Provider or Security Token Service to do the user authentication.
- **Service Provider (SP)** – This is a system entity that receives and accepts an authentication assertion issued by a SAML identity provider.

**Metadata**

A SAML metadata document describes a SAML deployment such as a SAML identity provider or a SAML service provider. Deployments share metadata to establish a baseline of trust and interoperability. When registering Service IDs for CAS/SAML authorization, providing accurate and compatible metadata from the service provider will be required.

**Authentication Flow**

Then following diagram describes the flow of authentication between a User agent (web browser), Service Provider (SP) and Identity Provider (IdP). This is how each entity interacts in a SSO session using SAML 2.0 ONLY.
UGA implementation

(Shibboleth is the SAML IdP for SSO)

SP owners can access UGA's metadata endpoint at https://sso.uga.edu/cas/idp/metadata

CAS can be integrated with the Shibboleth federated SSO platform by a couple different strategies. It is possible to designate CAS to serve as the authentication provider for the Shibboleth IdP. With such a setup, when user is routed to the IdP, the following may take place:

- If the user has already authenticated to CAS and has a valid CAS SSO session, the IdP will transparently perform the requested action, e.g. attribute release.
- If the user does not have a valid CAS SSO session, the user will be redirected to CAS and must authenticate before the IDP proceeds with the requested action.

Authentication Flow

The authentication flow is similar to that of the SAML 2.0 protocol with some notable differences. Note that this process is primarily dependent on HTTP redirects through the user's browser and that the authentication is delegated to the CAS server.

Rather than a user being authenticated by the IdP and then being redirected to the application, the user will be redirected to CAS and will need to be authenticated by CAS and then the IdP to gain access to the application.
Multifactor Authentication (Archpass)

Unless otherwise requested in the CAS request form, Service IDs will default to use multi-factor authentication using UGA’s ArchPass powered by Duo. After entering your MyID and password as usual, you will be asked to verify your identity with a device enrolled in ArchPass. You can find more information about ArchPass at Information Security’s webpage. Multi-factor authentication should be required when your site contains sensitive or restricted information. For more information about data classification, see Information Security’s webpage.

Sample Configuration

- PHP example

Disclaimer: The content on this page is intended to be used for informational purposes only. For more information on CAS, visit their protocol specification found here. For more information on SAML, visit theirs found here.

EITS does not own the CAS or SAML protocol.

Testing the Configuration

Test MyIDs can be checked out to test authentication while onboarding your application. If the application has special authorization criteria based on a local user base or attributes being passed, test accounts can be set up to mimic the specifications. Test account can be requested here: https://uga.teamdynamix.com/TDClient/Requests/ServiceDet?ID=30597

UGA Application Listserv

For applications that connect to UGA SSO, owners are automatically added to the UGASSOApps@listserv.uga.edu. This listserv is used for SSO communications about service maintenance, disruptions, or announcements. Application owners on this listserv can also use this for tips, techniques, and general questions about applications connecting to UGA SSO. This is list will be a discussion list and is not moderated before posts. By accepting your understanding of the authentication methods at UGA, you are also accepting, as an application owner, you will participate in the listserv communications.

UGA SSO Maintenance Windows

Information about UGA SSO and moving your application, please use the information found at https://eits.uga.edu/access_and_security/uga_sso/

Request for applications to go to UGA SSO will be processed through a workflow that passes between application owners, InfoSec, and IDM. Please plan at least 8 weeks, if no issues are identified, from initiation of request to the scheduling of projected application production go live date.

Development Environments

<table>
<thead>
<tr>
<th></th>
<th>Duration</th>
<th>Day</th>
<th>Time</th>
<th>Anticipated Service Disruption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Maintenance</td>
<td>8 hours</td>
<td>Any weekday</td>
<td>8 a.m. to 4 p.m.</td>
<td>Low - Medium</td>
</tr>
<tr>
<td>System Maintenance</td>
<td>8 hours</td>
<td>Any weekday</td>
<td>8 a.m. to 4 p.m.</td>
<td>High</td>
</tr>
</tbody>
</table>
### Stage Environments

<table>
<thead>
<tr>
<th>Service</th>
<th>Duration</th>
<th>Day</th>
<th>Time</th>
<th>Anticipated Service Disruption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Maintenance</td>
<td>2 hours</td>
<td>Monday through Friday</td>
<td>3 p.m. to 5 p.m.</td>
<td>Low - Medium</td>
</tr>
<tr>
<td>System Maintenance</td>
<td>8 hours</td>
<td>Last Wednesday of the month</td>
<td>8 a.m. to 4 p.m.</td>
<td>High</td>
</tr>
</tbody>
</table>

### Production Environments

<table>
<thead>
<tr>
<th>Service</th>
<th>Duration</th>
<th>Day</th>
<th>Time</th>
<th>Anticipated Service Disruption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Maintenance</td>
<td>4 hours</td>
<td>1st and 3rd Fridays of the month</td>
<td>11 p.m. to 3 a.m.</td>
<td>Low - Medium</td>
</tr>
<tr>
<td>System Maintenance</td>
<td>8 hours</td>
<td>1st Friday of the month</td>
<td>11 p.m. to 7 a.m.</td>
<td>High</td>
</tr>
</tbody>
</table>

Emergency maintenance will be conducted as needed. More information can be found at: [https://eits.uga.edu/access_and_security/uga_sso/moving_to_uga_sso/](https://eits.uga.edu/access_and_security/uga_sso/moving_to_uga_sso/)