The Federated Identity Management Cookbook

Erik Scott and Josh Drake
Contact: escott@renci.org

July 10, 2022

July 10, 2022
Version 1
CI Compass (https://ci-compass.org/) and Trusted CI (https://www.trustedci.org/)

Preferred citation: Erik Scott and Josh Drake, The Federated Identity Management Cookbook, version 1, July 10, 2002, DOI: 10.5281/zenodo.6815944

This work was supported by the National Science Foundation (NSF) Grant #2127548: CI Compass: An NSF Cyberinfrastructure (CI) Center of Excellence for Navigating the Major Facilities Data Lifecycle and NSF Grant #1920430: Trusted CI, the NSF Cybersecurity Center of Excellence.

Table of Contents

Identity Management: An Introduction

The Goal of this Cookbook

One of the basic tasks any Cyberinfrastructure (CI) has to handle is the problem of keeping track of who the users are, what they're allowed to do, and whether it's likely they are who they say they are. This is the core of Identity Management, usually abbreviated as IdM. At the simplest, it might be a list of users and passwords on a computer in the back of a lab. National Science Foundation (NSF) Major Facilities (MFs), on the other hand, may have many kinds of users in large numbers of subgroups, many overlapping, some mutually exclusive, and with complicated rulesets determining membership. The problem becomes worse when we face the challenge of applying those rulesets to a large number of data products, instruments, and CI resources. Managing this constellation of capabilities becomes time consuming and prone to errors with substantial impacts.

The goal of this document is twofold. First, we want to provide a few time-tested recipes for building IdM capabilities. This is indeed where our title comes from. Secondly, this report can serve as a quick introduction and primer on topics in IdM with references for further learning.

The Importance of IdM

Managing the CI for a Major Facility means, among many other things, keeping track of who is allowed to use your CI, what portions of it they are allowed to use, how they convince you they are who they say they are, and what to do about them when their circumstances change. Collectively, these tasks fall under Identity Management. It is not just usernames and passwords.

The first and last of those tasks - tracking who is allowed in and dealing with changes in that list - is absolutely as much of an administrative task as it is a technical one. IdM at Major Facilities is very much like it is at a large University. There are employees of the Facility, and they might be grouped into “staff” and “scientist” roles at some sites. There are visiting researchers and students, and there might even be some overlap (for example: staff members go to graduate school) to complicate things. Add in government users, contractors, and temporary access for vendors and it becomes more complicated still. This is an administrative chore. A fairly common solution is to take advantage of existing business practices and systems as much as you can. If you’re already using, just as an example, Peoplesoft to handle your HR, then having it automatically push out its view of the world into your IdM system is a good idea. There will still be special cases to handle manually, of course, but take advantage of automation as much as you can.

Verifying that someone is who they claim to be is called ***authentication*** and is sometimes abbreviated ***authN***. This is a vibrant area of research and practice. Techniques can be as simple as prompting for a username and a password, or you can ratchet up the complexity and add additional factors, such as physical tokens, biometric IDs, and trusted outside identity providers. When asking for multiple factor authentication (MFA) it is important to ask for different types of information for each factor such as something you know (a password), something you have (your phone or key-fob), or something you are (a fingerprint).

Conversely, deciding what portions of your CI should be accessible to a given user is known as ***authorization*** and will occasionally be abbreviated ***authZ***. At the authorization step, your CI has adequately determined the person’s username (the authN is finished) and now it is time to decide what actions they have permission to do. Complex authorization schemes are beyond the scope of this cookbook, and usually involve custom integration work between operating systems, standalone software, and web-based applications and services. The two most common authorization schemes are based on the user's role or attributes, and are referred to as Role Based Access Control (RBAC) or Attribute Based Access Control (ABAC). Most consider RBAC a subset of ABAC, as roles can be treated as identity attributes. A common starting point is to add attributes to your authorization system (see LDAP and ADFS later in the cookbook). This could take the conceptual form of “This is elscott4@ncsu.edu. CanUseLibrary = True. CanEnterReactorRoom = False” or to refer to each user by their role and grant permissions based on the access they will need to fulfill the duties of that role.

Distributed and Federated Systems

Cyberinfrastructure is universally composed of many computers on a fast network. The difference between a roomful of computers and a distributed system is that in the latter, the machines are cooperating and have some degree of dependency with each other. In distributed IdM, the computers are validating credentials

provided by users through some sort of exchange with the IdM servers. This is not a new thing at all. Unix had Yellow Pages for this since the late 80s and moved quickly to LDAP. The Microsoft world has had Active Directory for similarly long time to do the same things.

Federation is what you get when distributed systems cooperate across institutional boundaries. Technologically, this is very much like distributed systems. Sometimes allowances have to be made for network latency and limited bandwidth. Caching is a common technique for these problems. The hard part of the problem is the agreement and cooperation between institutions that is required. Each of the responsible parties has to be able to trust the policies and practices of the others and must also be able to verify adherence.

Multi-factor Authentication (MFA)

When we log into a system with a username and a password, we are authenticating with one factor— our password (our usernames are publicly known). Multi-factor Authentication (MFA) is just what it sounds like—logging in with a username and more than one way to prove we really are that person [1]. This is rapidly becoming essential. For instance, the NIH requires MFA to access their systems and your Federated IdM must be able to report compliance, and this notion is spreading. Increasingly, the evidence is convincing: passwords alone are inadequate. MFA is not a magic bullet and it will not magically solve all your security woes. The human using the keyboard is still the weakest link in the AuthN/AuthZ chain and is why Social Engineering, including Phishing, is so effective. Nonetheless, requiring some second form of proof dramatically improves your odds.

There are three types of things you can use to authenticate with—something you know, something you have, and something you are. Use of two items from different categories is good, use of all three offers quite a bit more security on top of that but at the possible expense of too much privacy and cost. The tradeoffs need to be weighed.

“Something you know” is a secret that can be shared with a system for authentication—practically always this is a password. Passwords are nicely portable but are sadly lacking in many other aspects. It is easy to forget a password, especially one you have not used in a while. People try to handle this by writing all their passwords down, but then they have created not only a high-value target to attack but also a single point of catastrophic failure. If that list of passwords gets lost then there is going to be a lot of work resetting everything.

“Something you have” refers to ownership of a physical device that has to be present and working to log in. Examples of this go back a long way. The RSA Token was ubiquitous in banking as far back as the early 90s. The device produced a new pseudorandom number every few seconds. When logging in, the user had to enter a username, password, and the current number displayed on the token. Because the device did not produce true randomness at all but rather stepped through a sufficiently complicated algorithm, it was possible for the login system to verify that the right number was entered.

Another example is the DoD Smart Card. This is a device that looks like a thick credit card and has electrical contacts on one end. It’s plugged into a computer where the user wants to log in. Inside the device is a small processor containing a unique key and running a cryptographic algorithm. The login system “challenges” the card with an input, the card cryptographically transforms that into the “response”, and the login system verifies that the encrypted response matches what should have been produced.

In the simplest case, the login system texts the user with a random number. The user then has to enter that number into the login system. The security of this is not great – it is very easy to hijack a cell phone number and have copies of all their texts sent to you. More sophisticated approaches either use the phone like an RSA Token or they respond to a message sent to them and run an authentication app. This app can force the user to enter the password for their lock screen, giving some reassurance that they know their account password, have knowledge of their cell phone lock screen password, and have their phone with them right then.

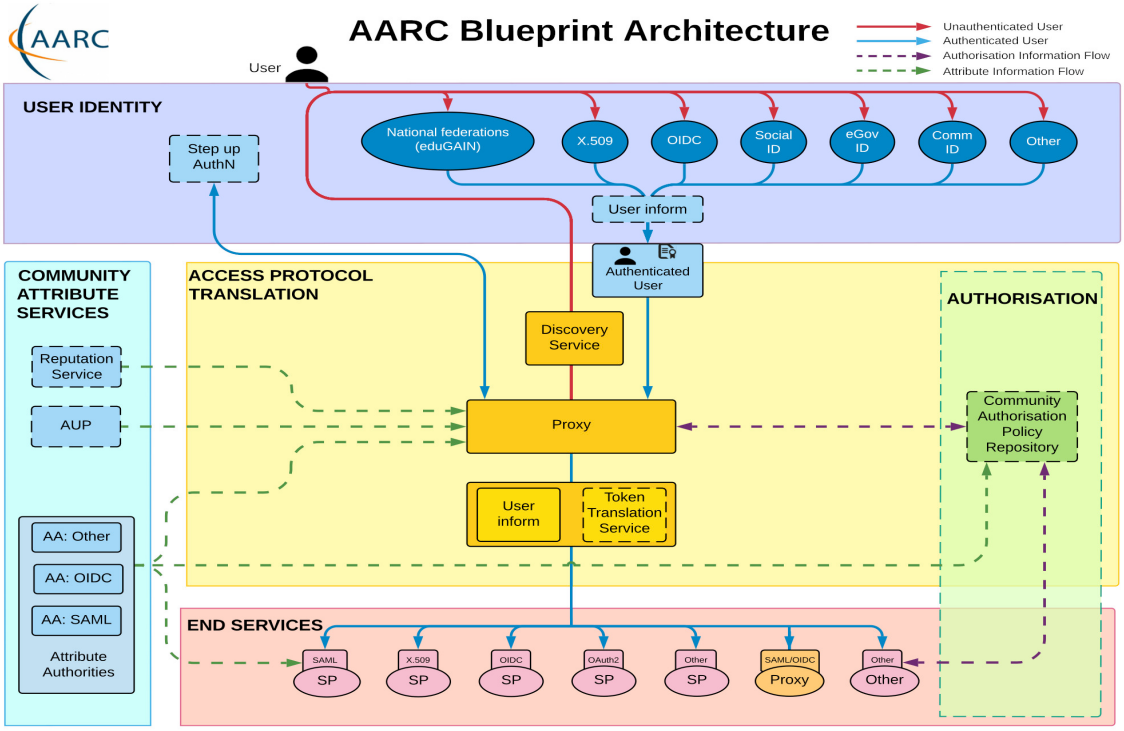
Finally, the "Something You Are" heading refers to the login system being able to measure something about you. Cell Phones now routinely offer to handle device authentication through fingerprint readers or facial recognition. It is worth mentioning that biometrics has a false positive and a false negative rate, and there is another problem, too: revocation. If your password gets compromised and shared then you can just change your password. If the machine representation of your fingerprint is leaked, there's nothing you can do to change it. It's also worth knowing that storing biometric information in your facility may cause privacy concerns for your users and could pose an alluring target for attack.

AARC Blueprint Architecture

The EU Horizon 2020 program launched the "Authentication and Authorization for Research and Collaboration" (AARC) initiative to facilitate greater cooperation and interoperation capabilities in the EU research community. AARC takes a multi-pronged approach, much like InCommon [AAD], in that they both facilitate community interaction to form standards and practices and also create technical work products. One of these products is the AARC Blueprint Architecture (BPA), a high-level requirements gathering and design study [2].

The Blueprint has proven highly influential in shaping IdM technologies and adoption. It does not specify algorithms or protocols, but rather it describes functions that need to be provided and lays out a high-level view of how the pieces fit together. The Blueprint does not specify particular software packages to use. In fact, attention is paid to being able to integrate many different software packages and types of providers. Conversely, the groups that develop IdM technologies have a strong incentive to ensure their products fit the BPA so they can address a very wide audience.

The Blueprint specifies five major component groups. The following description is taken directly from their own description of the BPA:



- User Identity: services which provide electronic identities that can be used by users participating in international research collaborations.
- Community Attribute Services: components related to managing and providing information (attributes) about users, such as community group memberships and roles, on top of the information that might be provided directly by the identity providers from the User Identity Layer.
- Access Protocol Translation: defines an administrative, policy and technical boundary between the internal/external services and resources.
- Authorization: contains elements to control the many ways users can access services and resources.
- End-services: where the external services interact with the other elements of the AAI [Authentication and Authorization Infrastructure].

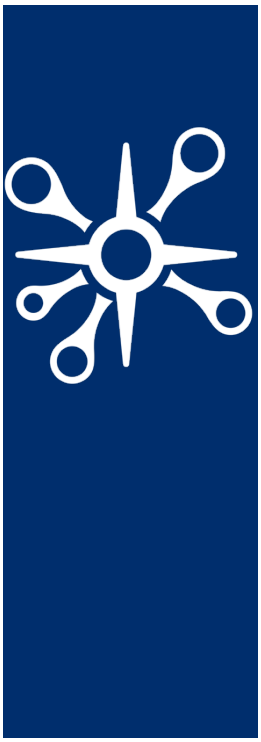
A “number of” pilot projects were launched resulting in 18 prototype implementations. Results from these experiments informed a large and helpful collection of wisdom in the “Guidelines” series of documents [2].

An interesting feature of the AARC efforts has been some beginning of standardization in Authorization. This is possible because of agreement on a useful set of common metadata, a richer set of information that the “bare minimum” required by InCommon. To be fair, InCommon predates the pilots and may well have shaped some of the requirements.

How to Use This Document

The Cookbook serves two purposes. First, it is an introduction to the topic of IdM suitable for rank novices but with enough information to serve as a reference and some quick pointers for experts. Feel free to cut and paste from here to answer the questions that seem to pop up over and over. Secondly, we have a quick reference guide for IdM protocols. The ways in which identity information is exchanged can be a real alphabet soup; this section serves as a glossary with links to more detail, often to canonical sources. The third section is a reference to common software tools, software vendors, and outsourced/cloud service providers. In the fourth section, the “Cookbook” chapter itself walks through a few common IdM scenarios and shows good practices for solution design. Finally, references are gathered in one place at the end, divided by section.

Identity Management Protocols



SAML

SAML (Security Assertion Markup Language) [4,5,6] is a structured protocol for authentication to web applications. SAML is XML based, and uses a similar structure of hierarchical and nested data structures to store information in a prescribed way. The purpose of a SAML token, or SAML assertion is to pass data between an Identity Provider and a Service Provider. This allows one organization to assert a users' identity in a token, which can be based to multiple different providers via browser redirects, allowing for Single Sign-On (SSO) for web-based services.

OpenID Connect (OIDC)

OpenID Connect (OIDC) [7,8] is an identity layer built on top of the OAuth2.0 framework, as described in the next section. OIDC is functionally similar to SAML, but uses JSON Web Tokens (JWTs) to pass identity assertions rather than a structured data markup language like SAML. When using OIDC, users can be redirected to an OIDC provider URI to authenticate, and then have their authenticated identity passed back to the website or service seamlessly. OIDC tokens use a standard JWT structure and include standard claims, specified by the OpenID Connect Specification [9]. Standard claims include information about a user such as a name, gender, email, birthdate, etc.

Both SAML and OIDC achieve the same things, but differ in methodology, technology and capacity/capabilities. The following list of differences is at least partially adapted from information on Auth0's website [10].

- **IDP / SP vs. OP / RP:** In both OpenID Connect and SAML, an application (called SP - "Service Provider') in the case of SAML and RP - "Relying Party" - in the case of OpenID redirects the user to the identity provider for authentication. The difference here is that SAML does not connect well with certain applications (such as mobile applications), as compared to OpenID that works well with both web-based and mobile applications.
- **Message Format:** In OIDC, we have a JSON Web Token (JWT) id-token which provides the authentication information. On the other hand, in SAML, we have assertions which represent the authentication, attribute and authorization statements and are formatted using XML. JWTs are light-weight as compared to heavy XML assertions.
- **Support for API:** OIDC came out of necessity—SAML is too heavy and cannot integrate well with APIs. OIDC works by using RESTful API communication that utilizes the HTTP communication channel to send light-weight JSON security tokens for the authentication process, whereas SAML uses SOAP, which is also a protocol layer over HTTP, but it sends heavy XML messages for user authentication.
- **Mobile-centric Authentication**: As mentioned above, OIDC uses RESTful communication to create lightweight JSON security tokens that are passed between IdP and Relying Party, this makes OIDC a forefront protocol for mobile centric application authentication, unlike SAML which was mainly developed to be used for web application authentication, and because it uses XML there is little or no future space for it to be used in mobile applications, giving an edge to OIDC being used exclusively in mobile applications.

- **User Consent:** Because OIDC is a layer placed upon the OAuth framework, OpenID Connect can provide a built-in layer of authorization, which prompts a user to first consent to what the service provider can access. Even though SAML can provide consent flow, it does this through hard-coding done by the developer, instead of having it as a standard in its protocol.
- **Static Authentication:** In SAML, static authentication is required where the IdP and the Relying Party must be configured to know each other before the actual transfer of information takes place. For OIDC, this is not the case.
- **Implementation:** OpenID is much more simple to implement and is developer friendly, which extends the use cases over which it can be used. It can also be implemented easily from scratch or using libraries available from OpenID [11] to assist developers.

Use Cases

Using OAuth 2.0 Framework in conjunction with a bearer token implementation such as OpenID Connect allows a resource provider to grant access to its resources based on the digital identity of its users once set up and configured with an Identity Provider like CILogon [15], GitHub, Auth0, and others.

One simple implementation might use CILogon as an Identity provider, and mod_auth_open_idc module on an Apache server to allow EduGain [16] and InCommon [2] federation users the ability to access web-based resources.

More complex implementations might include using COManage (discussed later) to validate and store attributes about federated user identities, writing a custom OIDC implementation in flask or python, or adapting an existing OIDC implementation like Globus [17].

OAuth 2.0

OAuth2 Overview

OAuth 2.0 is an authorization framework which enables a third-party application to obtain limited access to a resource via an HTTP/S service [12]. This is done either via an approval interaction orchestrated by the resource owner, or by a third party application that obtains access on its own behalf from an authorized source.

As opposed to a traditional client-server authorization interaction, the OAuth 2.0 framework enables authentication and authorization without requiring the resource provider to store passwords or handle password authentication, while only granting access to the resources requested.

The OAuth 2.0 framework addresses the issues in traditional challenge/response password-based authentication by adding an authorization layer to the interaction and separating the role of the client and the resource owner. In the OAuth 2.0 framework, the client obtains an access token (see below) which contains several attributes about the client, such as identity, scope of request, lifetime of granted access, etc. These access tokens can be handled by third party clients and are issued by an authorization server that is approved by the resource owner. Once granted, a client may use the access token to access whatever resources were requested for the lifetime of that token.

OAuth defines its interactions across four key roles:

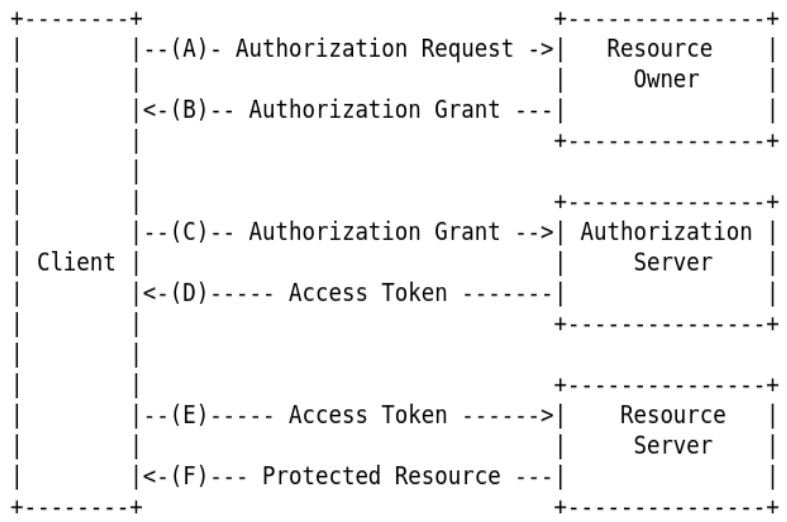
- *Resource Owner (RO):* a person or entity capable of granting access to a protected resource
- *Resource Server (RS):* the server hosting the protected resource and capable of accepting and responding to requests using access tokens

- *Client:* an application authorized to be making protected resource requests on behalf of an owner. Client does not imply particular implementation characteristics such as desktop, web based, server based or other devices/platforms
- *Authorization Server:* a server issuing access tokens to the client after a request and successful authentication of identity and authorization to grant access from the resource owner.

AAI Process

A typical OAuth 2.0 protocol flow may look like this:

1. The client makes a request to access a resource from the resource owner (RO). The request can be made directly if the client has already obtained an access token, or with an intermediary authorization server if they do not have a token.
2. The client receives an authorization grant, which is a credential bearing the RO's authorization (authZ) in one of four types (see https://datatracker.ietf.org/doc/html/rfc6749 Section 1.3 for more info).
3. The client requests an access token by using the authorization grant obtained from the RO in step 2.
4. The authZ server authenticates the client and validates the authZ grant, if both are valid, an access token is issued.
5. The client requests the protected resource from the resource server (RS) and authenticates using the access token issued in 4.
6. The RS validates the access token and serves the request for resources.



JSON Web Tokens (JWT)

JSON Web Tokens (JWT) is a way to encode claims that can then be signed. A token can carry information about its bearer (e.g. identity attributes) that allow a remote system to make authorization decisions without consulting with a central authority in real-time. Signing and verifying is used to ensure the data has not been modified. JWTs typically have a finite lifetime that is encoded in one of the fields. IANA defines a set of standard JWT 'claims' or fields [18], which can be customized as needed for a given system.

JWTs are not inherently part of OAuth 2.0, but since OAuth does not specify the authorization token format, JWTs are compatible, so JWTs are often used when federation is required. Token expiration is also part of the format. The website https://jwt.io/ is useful for looking at the format.

System for Cross-domain Identity Management (SCIM)

System for Cross-domain Identity Management, known as SCIM[13], is a protocol for exchanging IdM information between identity management systems. It was originally designed to simplify access to Cloud Software as a Service (SaaS), but it has proven useful for managing authentication tasks among numerous

kids of IdPs. The underlying protocol uses JSON and presents a REST interface, greatly easing the burden of integrating software with it.

Legacy

Old software never dies, it just becomes “Legacy”. Identity Management is no exception. IBM’s RACF running on mainframes is 46 years old and probably has a few more decades left in it.

LDAP

The Lightweight Directory Access Protocol (LDAP) [20] is a means for handling the lookup and verification of user data. Dating back to the ISO X.400 - X.500 era, LDAP is still widely used and development is active. OpenLDAP [21] is the most notable Open Source implementation. It’s worth considering whether LDAP or SCIM (above) is a better fit for your problem, and they’re not mutually exclusive.

Active Directory

Microsoft’s Active Directory (AD) [22] is their approximate analog of LDAP. Developed to replace the inflexible and horribly insecure NT LAN Manager package, AD performs the Domain Controller role in Microsoft’s network vision. It also serves as an LDAP server so both Windows and Linux resources can be consumers. AD users should consider moving to Azure Active Directory (below), a different solution altogether despite the confusingly similar name. Azure AD is the direction for future Microsoft development.

Samba

Samba [23] is an Open Source implementation of some of Microsoft’s important network protocols including the ones for file sharing (SMB or CIFS, names shifting over the years). Samba can use their “winbind” library to allow AD to control access to Samba resources. Older versions can be NT Lan Manager (NTLM) controllers, but this capability has been deprecated and in fact removed for many years now. NTLM security is trivial to bypass.

Identity Management Tools and Service Providers

Building with the Internet2 InCommon Stack

The Internet2 Consortium [24], besides ensuring the operation of a very fast network connecting research institutions and facilities, also works to facilitate interoperation of member institutions and their resources. This is done through facilitating common standards, and one of these thrusts is the InCommon Federation [2].

The federation, with over 700 participating institutions, agrees on standards in the Identity Management space for a distributed system capable of authenticating users and making a minimum baseline set of assertions about those users available to all the other members. These standards define the network protocols, but just as importantly they also describe the baseline operational security and organizational standards for the members. If Whatsamata U. asserts that a user is Jane Doe, that her email address is jdoe@whatsa.edu, and that she's a faculty member, then you can be reasonably certain that the statement is true and that she had to show an acceptable form of ID at some point before she even got her account.

In a stroke of good fortune, Internet2 along with other developers and institutions even provide open-source software to implement the InCommon protocols [25]. The protocols and corresponding software that we're most interested in are Shibboleth, COmanage, and Grouper.

Shibboleth

When you use a University library, there's a good chance you use Shibboleth to log in to their web site when you read a journal article. Shibboleth [26] is a software package that provides a Single Sign On (SSO) service for web and even standalone applications. It was originally developed by Internet2 and the spinoff Shibboleth Consortium was created to sustain the work into the future. Shibboleth implements the SAML and OIDC protocols discussed earlier.

Shibboleth has several things going for it. The biggest is simply providing the value inherent in Single Sign On. While it seems like "one username and password to rule them all" is dangerous, the reality is that it can be more secure than alternatives such as having dozens of passwords for dozens of resources and all of them written down in a notebook just begging to be stolen.

Another useful property of Shibboleth is that it can be optionally configured to allow the user control over how much information is shared between the Identity Provider and the Resource Owner. As a concrete example, when I log in to the NC State University Library to view journal articles with my NCSU username and password, I go through the Shibboleth username and password process. The next screen after that shows me what information the NCSU Authorization Server will present to the library's system. In my case, it indicates to me that it would share my username and my faculty/staff/student status. I have to explicitly tell Shibboleth that it's OK to share that information. Such control over information sharing is unusual in internet systems.

Sometimes people will confuse Shibboleth with Eduroam. It is an honest mistake - both handle identifying users to a system and both have some historical connection to Internet2. They are different, though, with Eduroam handling the use case of providing network access to visitors with a minimum of fuss versus Shibboleth's role as a general-purpose IdM approach.

It is worth noting one of the limitations of Shibboleth. Using Shibboleth to allow in users from only one institution is easy. Allowing users from a handful of places is a little harder. The technical work is not that bad, but it still has to be done. The hard part is satisfying the requirements of the other institutions so they can trust you, proving that you really did satisfy them, and having a means to quickly warn them if something goes wrong. After all, it is a really bad idea to trust IdM from a provider who is six months behind on their patches and does not really know if their users are who they say they are or not. It is easy to keep track of when you have a small number of federated institutions: the Triangle Research Libraries Network has four members, all close by, and their Shibboleth specialists all know each other. Doing this across a few thousand research universities and institutions is an entirely different matter, and this is where federations like InCommon are essential.

CoManage

Major Facilities are complex organizations and there's nothing to be done about that. At least there are some IdM tools available to help. CoManage [27] is an application that lets administrators create groups of users and assign identifiers that describe what they're allowed to do. A web services API lets the components be "pluggable", i.e. written separately as independent modules that run as though they're part of CoManage itself. Along with basic authorization based on group memberships, CoManage also handles some administrative chores for you. It can, for instance, maintain mailing lists based on group memberships and it can keep track of accounts that are linked to the same person (scotte@unc.edu is the same person as escott@renci.org). In fact, part of the suite is a tool to help automate that task.

Grouper

The Grouper [28] software simplifies the management of… groups… when there are complex rulesets involved. Major Facilities can be as complicated as a university. There can be thousands of employees in a combinatorically explosive brew of classifications, projects, sites, and countries. Add to that all the various kinds of visiting scientists and staff, students from K-12 outreach to Postdocs, and time-limited appointments. That barely scratches the surface. Grouper is made to handle large rulesets and make group assignments simpler, even detecting conflicting statuses.

CILogon

Shibboleth was originally designed to manage access to resources when the web site that provides the information and the users who want access are all part of the same organization. InCommon is, in effect, an agreement between several thousand institutions to adhere to specific standards for IdM so they can trust each other's IdM assertions - in other words, one username at your home institution is good enough to log in and use a resource at a totally different organization across the country from you. All this is predicated on the far away organization placing some entry in their authorization system saying, in effect, "that specific user at that specific institution is explicitly permitted to access this resource". Just because something is using InCommon does not mean it has to allow access to the whole academic internet.

This federation is great until you realize that in the most general case, several thousand institutions would have to manage trust relationships between each other in what becomes an N2 problem. Imagine your Major Facility having to continually satisfy several thousand other institutions' security auditing. That would probably be more than one full time job.

CILogon2 [14] was developed to solve this problem through two steps. First, the operators of CILogon handle the day-to-day operation. You do not install CILogon and run it yourself - this is Software as a Service (SaaS) and it frees the MF from a substantial workload. Secondly, CILogon manages maintaining the mutual trust relationships of the InCommon members, turning an N2 problem into an easy one (from your standpoint). Quoting CILogon's website,

CILogon enables researchers to log on to cyberinfrastructure (CI). CILogon provides an integrated open source identity and access management platform for research collaborations, combining federated identity management (Shibboleth, InCommon) with collaborative organization management (COmanage). Federated identity management enables researchers to use their home organization identities to access research applications, rather than requiring yet another username and password to log on. Collaborative organization management enables research projects to define user groups for authorization to collaboration platforms (e.g., wikis, mailing lists, and domain applications). CILogon implements the AARC Blueprint Architecture.

CILogon has a substantial history, with the original paper published in 2014 [29]. Over time, CILogon grew into CILogon2, and along the way it gained tight integration with CoManage (above). Generally, the CILogon name refers to CILogon2, even in CILogon's own documentation. Using CILogon is, without question, the best way to take advantage of the entire Internet2 Identity and Access ("IAM") software stack and makes Shibboleth and CoManage integration remarkably simple. The principal way to use CILogon for new development projects is to treat it as an OIDC provider. For more information on this, see the Cookbook recipe for "Managing Access to Web-Based Resources using Federated ID", and also see the OIDC entry in the "Protocols" section. For non-web projects, CILogon can generate X.509 certificates (see the entry in this Cookbook) which can be used with a very wide range of CI categories.

Multifactor Authentication Solutions

MFA with Smartphone Apps (Duo Mobile, Google Authenticator, and LastPass Application Client)

The Duo, Google Authenticator, and LastPass smartphone apps are Multifactor Authentication (MFA) applications that turn the user's phone into an MFA token, so the phone is a "something you have" authentication factor. The workflow for all three is mostly the same. For Duo and Google Authenticator, the application running on the phone can produce a new number a few times per minute (a "token") or it can act in "push" mode where it waits for your IdM to send it a message. When used as a cellphone-based replacement for old style hardware tokens, a new number is produced at intervals. This number has to be entered into your IdM system. It is a lot like having two passwords instead of only one, this extra password is constantly changing, and the algorithm and starting seed for password generator is very, very difficult to determine.

All three of these vendors also have a mode where the phone just sits there and waits for a "push" notification from your IdM software. What happens is that when a user tries to log in, IdM sends a push to the user's cellphone. A dialog pops up on the phone when this happens. The user has to click "accept" and enter the lock screen password for the phone. This way, IdM can confirm that the user knew their password and (most likely) had their cell phone with them and knew the lock code for it.

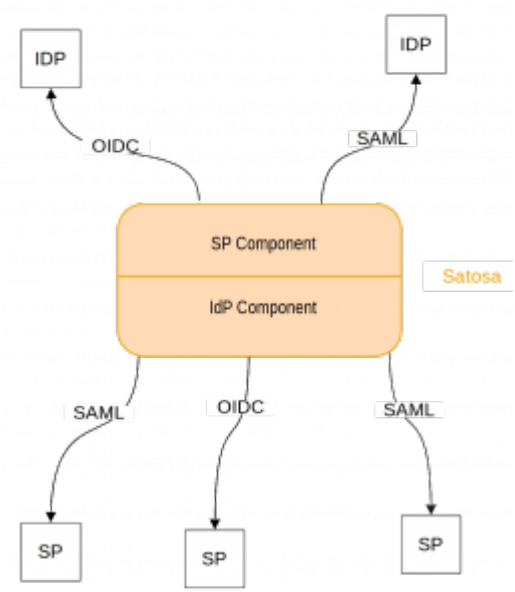
The problem with the "Cell phone as MFA Token" scheme is that it assumes the user remembered to take their phone with them, the phone works, and the phone is in an area with wireless data service. The latter issue is a problem with remote areas, which are also the best places to put things like big instruments, optical telescopes, or sensor arrays for ecology.

The workaround for the cell phone problem is the generation of a few "one time" access codes. These multifactor authentication applications can be configured to allow the user to produce a small quantity of numbers, usually 10 or so, that are in the roughly 10-digit range. If the push notification is not going to work for whatever reason, users can root around in their belongings and dig out their list of one-time codes. When one of these numbers is used to authenticate, it gets crossed off the list. While this scheme does make it possible to log in in the face of a dead phone, for example, it puts us right back to having passwords written on slips of paper in easy-to-steal places. This is why availability of this is configurable.

Yubico

Yubico has sold several generations of hardware “keys” for authentication. The devices have been in a conveniently small “thumb drive” form factor. They slot into a USB port and can mimic either a keyboard or a DoD-style (though not actually DoD compatible) Smart Card reader. The newest revisions offer Near-Field Communications (NFC) and can do their cryptographic exchange at fairly close distances to a laptop or other computer with an NFC reader. The Yubico keys are very frequently configured to act as Challenge-Response devices. In this mode, the IdM software presents a small amount of information to the key, which in turn does a “conventional” public key authentication.

Satosa



Satosa is a proxy software for Single Sign-On (SSO) [30,31,32]. It stands for SAML to SAML proxy, and allows for sites using AARC blueprint architecture to share identity information with one another using SAML assertions. Satosa operates very similarly to Shibboleth, in that it acts as a proxy between two entities: an IDP (Identity Provider) and a SP (Service Provider). In addition to forwarding SAML assertions, Satosa can alter the assertions by changing or adding attributes from external databases, including switching which authentication protocol is being used. While many protocols are supported, the most common ones that are supported out of the box are SAML and OIDC.

Because Satosa is a proxy server it can work in the middle of multiple systems using different protocols, allowing disparate systems to share digital identity assertions with one another between platforms.

In growing or highly changeable environments, Satosa can allow for the changing or components or addition of new protocols/systems with little expense, as new protocols can be handled using new software integrations. For instance, the addition of an LDAP based registry can be handled with a new interface on the LDAP side and can output to existing protocols already in use with minimal development effort.

ADFS (Microsoft)

Microsoft Active Directory Federation Services (ADFS) [33] is an addition to AD that allows the production of credential certificates that can be used across trust domains. ADFS supports SAML and with some work Shibboleth can be used as a proxy to it. This would make Shibboleth one of the user-facing components, and ADFS is one of the federation servers that are sometimes used with InCommon federation.

If the Major Facility is already heavily invested in Microsoft’s directory services, has plenty of expertise in that area, and is short on Linux/Unix expertise, then they should evaluate Azure Active Directory. If Azure AD is not a workable path then ADFS could be a reasonable (albeit potentially end-of-life) way to go. Bear in mind that, at present, ADFS’s support of MFA is an all-or-nothing affair and there’s no way for an external site to indicate that a higher assurance level (say, AAL2) is required. This factor (no pun intended) is probably enough to sway the balance toward Azure AD.

Commercial Identity Providers (IdPs)

Many of the largest web companies provide some mechanism to let other web sites piggyback on to their IdM. Of course, this means those companies have even more information about us now, but users seem to enjoy the convenience of having fewer usernames and passwords to remember - most of the point behind Federated ID. While ORCID is a superior way to identify individuals from institutions that do not participate in InCommon, commercial IdM providers can also be a way of handling these users.

In this space, Google and Facebook are the most obvious Identity Providers (IdPs). Google and (of course) ORCID are supported by CILogon, for instance. GitHub identities are often seen for services that are more oriented toward software developers. Microsoft IDs (such as ones for OneDrive) are another alternative. Despite Microsoft having bought GitHub, the two sets of users have not been merged.

Writing the code and managing access keys for multiple Identity Providers is a time-consuming affair. Commercial providers have emerged that provide the software and the operational support for federating identities from these providers. The biggest player among these is “Auth0” - that’s a zero at the end, and be careful to not confuse it with OAuth.

Auth0

Auth0 [35] is a cloud-based, commercial IdM provider, offering one-stop integration for multiple identity providers (“over 40” as this is being written) in an environment that they host. Their environment is easy to configure and operate. They offer a web-based administration console, for instance. An OIDC interface is presented to your applications and systems. One of the strong points of Auth0 is support for machine-to-machine authentication.

Auth0 targets the commercial web development space, unlike CILogon or the InCommon stack (above) which is principally used in the Research and Education field. The service presents a nicely integrated set of services and handles tasks such as user signup, brute-force attack detection, connection to Multifactor Authentication methods, and some basic analysis of user logins.

Azure (IdM as a Service)

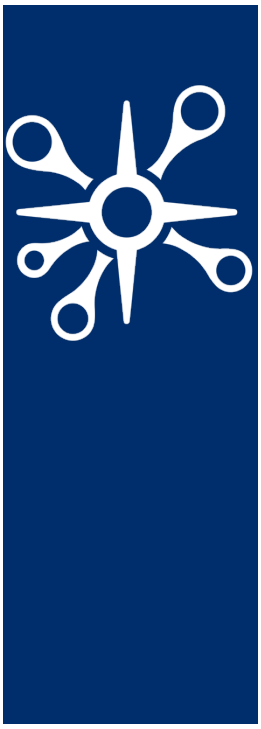
Microsoft’s Azure Cloud provides rental access to scores of services and not just virtual machine rental. One of the services they will sell is outsourced Identity Management through their Azure Active Directory (AAD) service [36]. There are four tiers of service available [34]. The cheapest is “free”, in that it comes bundled with Office365. This offering is hosted in Microsoft’s Azure Cloud, of course, but is more than just “renting a VM and installing AD”. Azure AD, in fact, goes beyond what on-premises AD offers and will be the target of Microsoft’s future IdM efforts. With AAD, Microsoft manages the day-to-day operation of the server software. This can save a little bit of labor cost, but more importantly the operation of the software is being done by a (presumably) well-trained individual. Given the complexity of managing any IdM system, not just AD, and the security implications of getting it wrong, using an “IdM as a Service” offering might be a good thing for your facility to consider. The disadvantage, though, is that you have no control over what Microsoft chooses to do.

PingIdentity

PingIdentity [37] is a commercial software and cloud services company selling a suite of Identity and Access Management (IAM) tools. Most interesting in the context of this Cookbook is PingFederate. This package bridges whatever IdM software you’re using so that the outside world can federate with it. Protocols supported include SAML, OAuth, and OIDC. This software can be made to interoperate with the InCommon

Federation stack, given sufficient time and effort. For organizations that are “cloud native”, the cloud hosted version of this product could possibly fill a need.

The Cookbook: Time-Saving Recipes



Managing Access to Web Based Resources using Federated ID - CILogon and OIDC

If you are building or upgrading CI, you are almost certainly going to provide at least some of the access through a web user interface. There is no shortage of ways to do this. In this recipe we will use the Python Django framework as our example. The IdM strategy will be to use CILogon. This is a common approach used by several Major Facilities, organizations and CI projects, and is as well as a versatile one.

Our example here will be a very simple application we originally developed as a demonstration for the NOIRLab Major Facility. It allows newly appointed visiting scientists to log in to a Django application using their home institution credentials, fill in a blank with their laptop's wireless MAC address (for network access), and log out. The MAC address is saved in the Django instance's Postgres database.

Key to being able to handle guests and visiting appointments from many home institutions is the use of InCommon and CILogon. InCommon, covered in more detail elsewhere in this Cookbook, is the organizational framework that allows a great many universities and organizations in the US (and by agreement, the rest of the world) to share identity management assertions. CILogon is a software system that makes taking advantage of that federation relatively easy.

We will develop this demonstration in three steps. First, we will build it completely absent of authentication, just to serve as the most minimal Django application we can make (or that will still record data, anyway). The second step will be to refactor the application a little bit and enable Django's built-in authentication and authorization (AA). Using this built-in AA is the naive approach - usernames and passwords have to be maintained by hand. Finally, we'll add support for handling logins via CI Logon and by installing the mozilla-django-oidc package and letting it subclass (essentially "replace") the built-in Django authentication mechanism.

To work through this example, you will need:

1. A CILogon development account, configured for your organization and project. For information see www.cilogon.org .
2. A working instance of Django on Linux or MacOS. This example assumes you are using Linux.
3. A copy of the source code for the demonstration application from https://github.com/ci-compass/CoManageForDjango3 .

Let's get started. It'll be fun!

The Barebones Minimal App

The “macregistry” directory contains what is almost the simplest possible Django application that allows for data entry and recording. This guide is definitely not an introduction to Django. For a reasonably quick introduction I recommend

https://www.youtube.com/watch?v=SlHBNXW1rTk&list=PLEsfXFp6DpzRMby_cSoWTFw8zaMdTEXgL

With that said, let’s take a quick look.

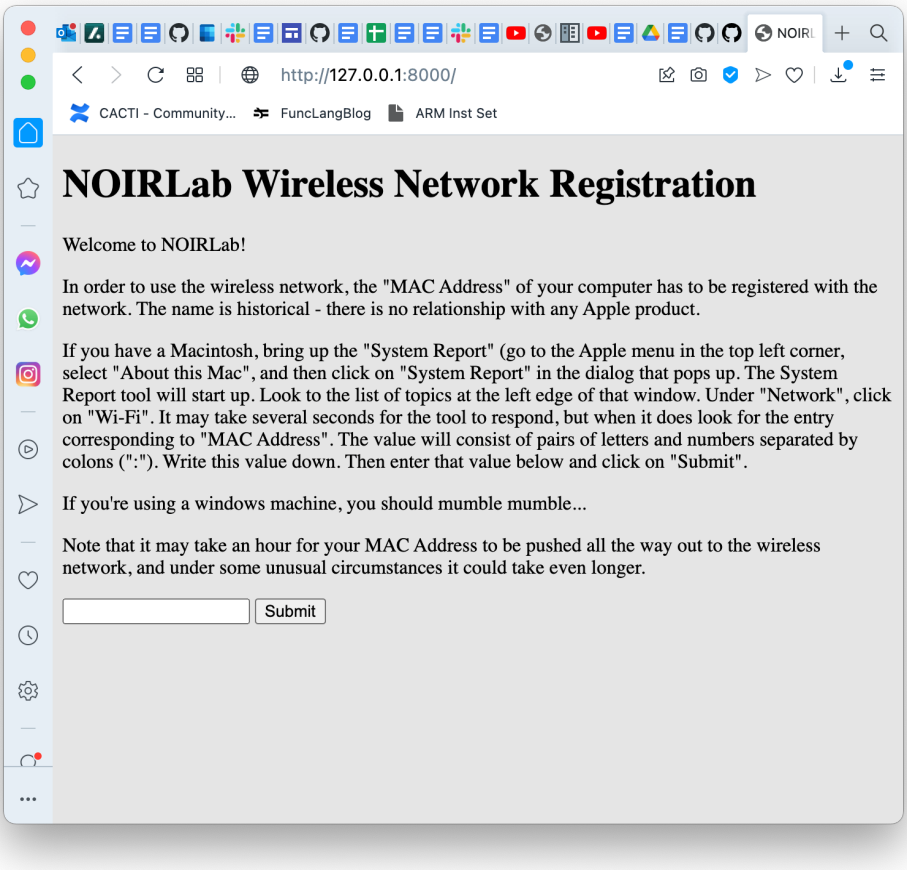
At the top level of this application we see some of the usual suspects. The db.sqlite3 file is a dead giveaway that we’re using sqlite3 instead of postgres so far. The manage.py and pyenv.cfg files are created by Django’s built in startproject command, as is the noirlab_comanage_demo directory. Forgive the name for the time being - this is still the non-authenticated version we are talking about. This directory contains the main body of the application. Note also the “macaddrs” directory. This is made for us by the “startapp” command and it contains the dataclasses structure for recording a list of MAC addresses. Finally, the templates directory holds the basic outline of the HTML that we’ll be generating.

Looking down into the noirlab_comanage_demo directory, there are three files that were modified from their automatically generated origins: settings.py, urls.py, and views.py. In urls.py we can see that Django is going to serve three URL paths. These are the root level, /register, and /admin. The admin hierarchy is created automatically for us and we can ignore that. The /register path receives the data and saves it in as an instance of a macaddr object. The root URL simply shows a welcome message and provides a blank to enter our MAC address into.

Note that so far there is not even a way to specify our username. This truly is a barebones application.

The code to produce these pages lives in views.py and in macaddrs/views.py. These provide the home_view() and registration_view() methods, respectively.

Running the application–python manage.py runserver–will start the demo program running inside a very basic, tiny web server. It’s not a good idea to use this in production, since it’s really just designed for testing and debugging. Once the web server is running, you can point a browser at http://127.0.0.1:8000/ and you’ll see the application:



When you enter your MAC address and submit the form, the information is sent to the /register page using the GET method. That information is saved as a macaddr object, using python dataclasses, and ultimately is saved by sqlite3.

Adding Django's Built-In Authentication

Now that we have a very basic web application running in Django, we can add a way for users to actually log in. Django provides a basic framework for handling this, the "django.contrib.auth" package. This package provides hooks into the admin interface for managing user accounts and makes it fairly easy for your own code to interface to it. There is also a command "python manage.py createsuperuser" to create a privileged user who can administer other users.

The code has been refactored a bit here. The root path, presenting the first page the user sees, now puts up a username and password box. The user fills these values in and then clicks the "login" button. This sends the username and password on to the login view where these two values are extracted from the POST request. Do not worry—in production this would be secured by installing certificates and using HTTPS, but for the moment this is fine for educational purposes.

Inside the login_view() handler, you'll notice a call to "authenticate()." This checks the username and password. If they're valid, a "user" object is returned, otherwise the "None" value is returned. After checking to make sure there really is a user object, the program calls "login()." That sets up all the apparatus in the background to actually make Django treat the user as someone who has successfully logged in. Importantly, the administration of all this is done through the "/admin" URL, provided to us by Django without any work on our part.

Federated Login at Last: mozilla-django-oidc

The final example program is where we integrate the mozilla-django-oidc package, letting it take over selected parts of the authentication process from Django while leaving others alone. This is how we can add federated login via CILogon without *too* much extra work. Much of the following is derived from the mozilla-django-oidc installation documentation, with some changes made to reflect specifics of CILogon.

After installation, you willl need to do some things to get your site using `mozilla-django-oidc`.

Acquire a client id and client secret

Before you can configure your application, you need to set up a client with an OpenID Connect provider (OP). You'll need to set up a *different client* for every environment you have for your site. For example, if your site has a -dev, -stage, and -prod environments, each of those has a different hostname and thus you need to set up a separate client for each one.

You need to provide your OpenID Connect provider (OP) the callback URL for your site. The URL path for the callback URL is `/oidc/callback/`.

Here are examples of callback URLs:

- http://127.0.0.1:8000/oidc/callback/ -- for local development
- https://myapp-dev.example.com/oidc/callback/ -- dev environment for myapp
- https://myapp.herokuapps.com/oidc/callback/ -- my app running on Heroku

As a concrete example, we will now walk through setting up a project to use CILogon. We'll show some "reasonable" values for variables and configuration settings. Keep in mind your local site and project is going to require some changes, but we'll point those out when they crop up.

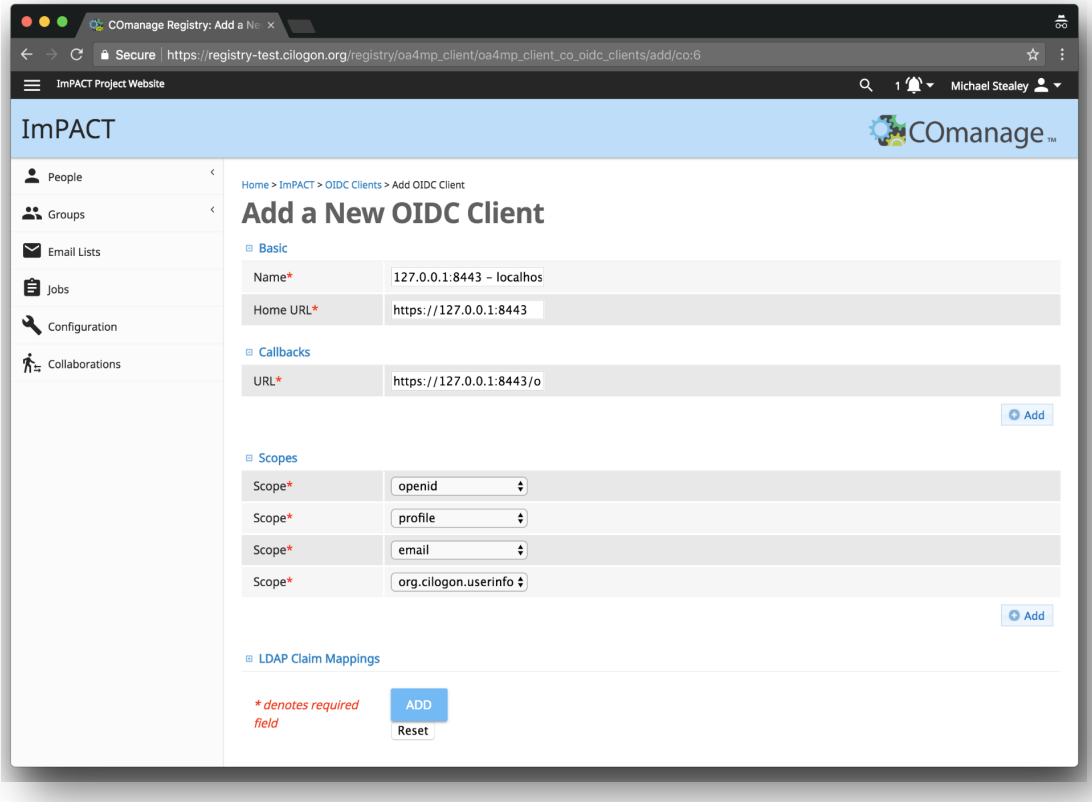
Create a COmanage OIDC client to act as the Relying Party (RP) to your Django based client.

Login to your COmanage registry: https://registry-test.cilogon.org/registry/

COmanage Registry access, by the way, is available on paid CILogon subscription plans (https://www.cilogon.org/subscribe). The no cost alternative is to use https://cilogon.org/oauth2/register for registering the OIDC application.

Choose to make a new OIDC client with the following fields.

- Name: `127.0.0.1:8443 - localhost`
- Home: `https://127.0.0.1:8443`
- Callback: `https://127.0.0.1:8443/oidc/callback/`
- Scope: `openid`
- Scope: `profile`
- Scope: `email`
- Scope: `org.cilogon.userinfo`
- LDAP Server URL: `default value`
- LDAP Bind DN: `default value`
- LDAP Bind Password: `default value`
- LDAP Search Base DN: `default value`



The OpenID Connect provider (OP) will then give you the following:

1. a client id (`OIDC_RP_CLIENT_ID`)
2. a client secret (`OIDC_RP_CLIENT_SECRET`)

You'll need these values for settings. Save them very safely and securely, because they'll only be shown to you once. If you lose them or if you leak them, your old setup will have to be deleted and a new one created. Be certain you never check those keys into source control, either. You can use your .gitignore file, for instance, to keep them out of GitHub.

Choose the appropriate algorithm

Depending on your OpenID Connect provider (OP) you might need to change the default signing algorithm from `HS256` to `RS256` by setting the `OIDC_RP_SIGN_ALGO` value accordingly.

For `RS256` algorithm to work, you need to set either the OP signing key or the OP JWKS Endpoint.

The corresponding settings values are:

```
OIDC_RP_IDP_SIGN_KEY = "<OP signing key in PEM or DER format>"
OIDC_OP_JWKS_ENDPOINT = "<URL of the OIDC OP jwks endpoint>"
```

If both specified, the key takes precedence.

In the case of CILogon, you'll need to use RS256 and you can set the JWKS Endpoint to `https://cilogon.org/oauth2/certs`.

Add settings to settings.py

Start by making the following changes to your `settings.py` file.

```
# Add 'mozilla_django_oidc' to INSTALLED_APPS
INSTALLED_APPS = (
    # ...
    'django.contrib.auth',
    'mozilla_django_oidc',  # Load after auth
    # ...
)

# Add 'mozilla_django_oidc' authentication backend
AUTHENTICATION_BACKENDS = (
    'mozilla_django_oidc.auth.OIDCAuthenticationBackend',
    # ...
)
```

You also need to configure some OpenID Connect related settings too.

These values come from your OpenID Connect provider (OP).

```
OIDC_RP_CLIENT_ID = os.environ['OIDC_RP_CLIENT_ID']
OIDC_RP_CLIENT_SECRET = os.environ['OIDC_RP_CLIENT_SECRET']
```

Warning

The OpenID Connect provider (OP) provided client id and secret are secret values.

DON'T check them into version control--pull them in from the environment.

If you ever accidentally check them into version control, contact your OpenID Connect provider (OP) as soon as you can, disable that set of client id and secret, and generate a new set.

These values are specific to your OpenID Connect provider (OP)--consult their documentation for the appropriate values.

OIDC_OP_AUTHORIZATION_ENDPOINT = "<URL of the OIDC OP authorization endpoint>"

OIDC_OP_TOKEN_ENDPOINT = "<URL of the OIDC OP token endpoint>"

OIDC_OP_USER_ENDPOINT = "<URL of the OIDC OP userinfo endpoint>"

Warning

Don't use Django's cookie-based sessions because they might open you up to replay attacks.

You can find more info about cookie-based sessions in Django's documentation.

Since we're using CILogon, we can use these values:

```
OIDC_OP_AUTHORIZATION_ENDPOINT = 'https://cilogon.org/authorize'
OIDC_OP_TOKEN_ENDPOINT = 'https://cilogon.org/oauth2/token'
OIDC_OP_USER_ENDPOINT = 'https://cilogon.org/oauth2/userinfo'
```

These values relate to your site.

```
LOGIN_REDIRECT_URL = "<URL path to redirect to after login>"
LOGOUT_REDIRECT_URL = "<URL path to redirect to after logout>"
```

For this example, it's fine to just redirect to the top level of our website. Use "/" as the value of those.

Add routing to urls.py

Next, edit your `urls.py` and add the following:

```
from django.urls import path, include

urlpatterns = [
    # ...
    path('oidc/', include('mozilla_django_oidc.urls')),
    # ...
]
```

Enable login and logout functionality in templates

Then you need to add the login link and the logout form to your templates. The views are `oidc_authentication_init`, `oidc_logout`.

Django templates example:

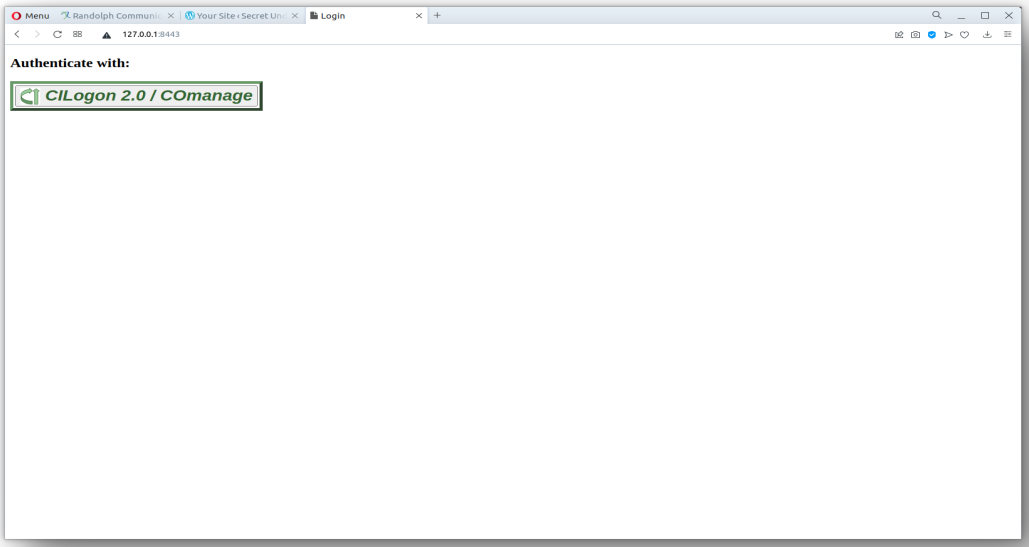
```
{% if user.is_authenticated %}
  <p>Current user: {{ user.email }}</p>
  <form action="{% url 'oidc_logout' %}" method="post">
    {% csrf_token %}
    <input type="submit" value="logout">
  </form>
{% else %}
  <a href="{% url 'oidc_authentication_init' %}">Login</a>
{% endif %}
```

Jinja2 templates example:

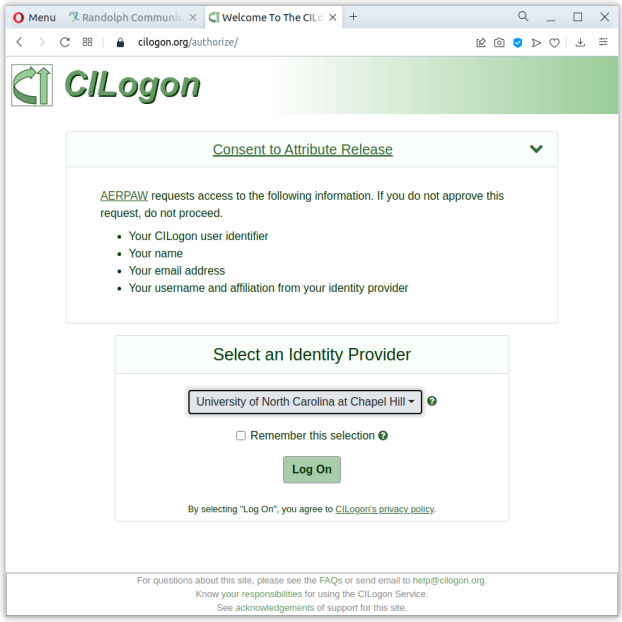
```
{% if request.user.is_authenticated %}
  <p>Current user: {{ request.user.email }}</p>
  <form action="{{ url('oidc_logout') }}" method="post">
```

```
    {{ csrf_input }}
    <input type="submit" value="logout">
  </form>
{% else %}
  <a href="{{ url('oidc_authentication_init') }}">Login</a>
{% endif %}
```

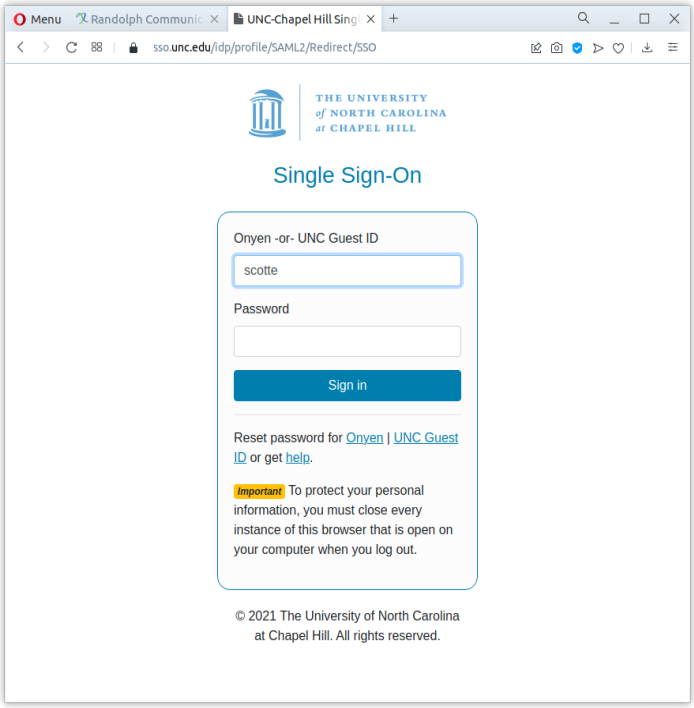
These are the changes made to the previous version of the application. When it runs now, we're presented with the login procedure:



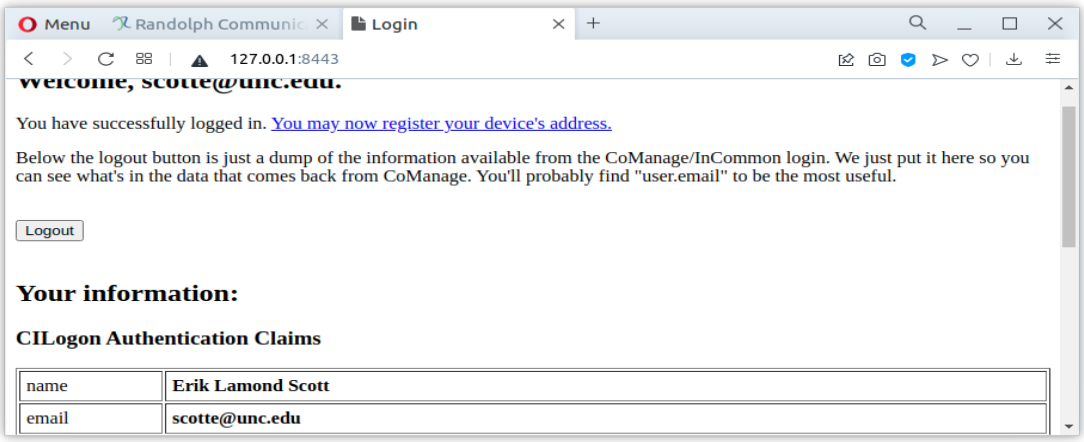
Clicking through takes us to the CILogon institution selection page. There are a few thousand options in the pulldown. Start typing a few characters of a unique substring to quickly find the one you're looking for. I type "chapel" and it zeroes in on UNC-Chapel Hill.



Clicking through to log on takes me to UNC's Shibboleth page:



And through the magic of HTTP Redirects, the successful login takes us to the page we specified in LOGIN_REDIRECT_URL: (note the URL shown - the top level of the local development server in this case)



It sounds like a lot but think of the amount of benefit we get from 20 or so lines of code. We have a smooth way to handle users from a few thousand institutions, cleanly integrated into Django's existing authentication module, and all using code maintained by someone else.

This example, by the way, assumes that anyone with a login anywhere is allowed to use it. It will record who they logged in as, of course, and it even goes so far as to populate the Django user objects and update the underlying database. This is not always what we want. If we need explicit control of who can use the application then we should either maintain a user list through Django itself or else we should read the user object created when we come back to the LOGIN_REDIRECT_URL and determine if it's an authorized user.

Allowing Access to Non-Web Systems and Applications Using Federated ID

Most new application development, at least publicly-facing applications anyway, have web based front ends. There are still a lot of places where important cyberinfrastructure does not have a web front end, though, and federated identity management is beneficial. Whether you want to graft a modern IdM approach onto a legacy application or just build new applications quickly and inexpensively, there are several good ways to let advanced IdM take some of the tedious work out of your life.

The simplest problem is handling logins for desktop computing and servers. The traditional solutions here are Active Directory for cases where people need to log in to Windows machines and LDAP for handling Linux (or other Unixes). These methods are fine for what they are meant to be, which is authentication inside of a single Trust Domain (essentially, a single institution). For handling federated login, there are some alternatives.

A very common approach is to use a federated IdM package such as CILogon and its interface to CoManage to issue and store Oauth tokens or, historically, ssh keys. The user authenticates with CILogon and requests a key. This key is downloaded, typically as a .pem file, and that key is used as the authentication to other resources as needed. In this case, the key serves as the authentication for otherwise-passwordless ssh.

Some existing applications are unable to use certificates and must instead rely on conventional Unix/Linux authentication. In this case, the Pluggable Authentication Modules (PAM) facility can be used. CILogon, for instance, has recently added support for the federated login instance to “push” LDAP records into your internal LDAP server. This could be, for example, OpenLDAP or Microsoft’s Active Directory (AD).

Standalone applications that need to have users log in to them can be modified to handle the appropriate over-the-wire protocols. For instance, the “shib-http-client” package, developed by the DARIAH-DE project in the EU, is a relatively minimal Java library to connect to a Shibboleth server and authenticate. Since it expects Shibboleth, it expects to receive SAML from the authenticator. Other packages are available for OIDC in a range of languages from Perl to Python to C [11].

Becoming an Identity Provider in a Federation

Congratulations on your facility deciding to join the InCommon Federation! There are a lot of steps to go through, but none of them are terribly difficult and most are common sense. There is a good set of documentation from Internet 2/InCommon and an active discussion group on their wiki to help you along.

First, you should go through the administrative steps outlined in https://incommon.org/federation/federation-join/. This will describe how to join the federation and has a link to the Participation Agreement. The agreement lists out the requirements your institution has to meet.

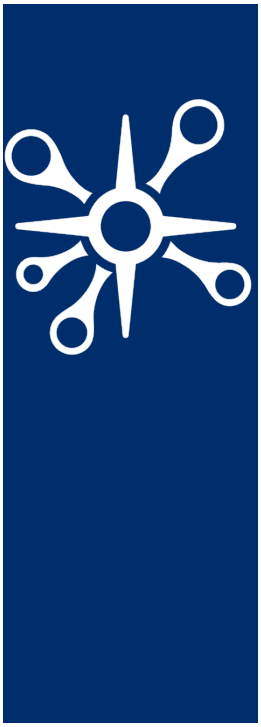
Once you’ve satisfied the baseline requirements outlined in the agreement, you have a decision to make: whether to operate your own SAML IdP interface software or to outsource the task. The answer is not

necessarily obvious. InCommon has prepared a guide to help with that decision—https://spaces.at.internet2.edu/display/TI/TI.106.1. It considers the size of an institution's IT department, its existing technical capabilities, and the infrastructure that is already in place.

If your decision is to run your own IdM stack locally, you need to decide on which software to use. Shibboleth is the dominant implementation in this domain with 90+% of institutions using it. SimpleSAMLphp [38] and Microsoft's Active Directory Federation Services (ADFS) account for most of the rest. Installing and operating Shibboleth has been greatly simplified with InCommon's release of a containerized version that runs in Docker.

Before you can call the job done, you need to update your InCommon Metadata. This is information about your institution, your IdP software, your administrative practices around identity, and contact information for normal business and for emergencies. Supplying the information and keeping it updated are crucial to the web of trust that makes InCommon work.

Appendix A: Other Topics in Identity Management



Identity Assurance

The biggest problem with Federated IdM is not technical. The hard part is knowing who to trust and to what degree. As we discussed in the section on InCommon, the creation, implementation, and monitoring of compliance with a baseline set of operational standards for IdM is a difficult but essential task. The Internet2 InCommon suite of practices, protocols, and standards is an excellent example in the research and academic space [2]. Other domains have their own requirements. Banks, for instance, have to meet FDIC Bank Readiness standards. The Federal Government has to adhere to the standards set forth in NIST Special Publication 800-63-3 [39].

Given the environment that NSF-funded MFs operate in, adherence to the InCommon Baseline Expectations is essential, even if you do not want to join the InCommon Federation and provide your facility's staff with the ability to use their MF credentials at other institutions. The Expectations are really just good practice, things like "your metadata must be accurate" and "we won't federate with your system if you don't trust it enough to use it yourself".

NIST 800-63-3 is a little bit more explicit. Reading that document, you will immediately see three terms in the executive summary, each with three levels of control. IAL, or Identity Assurance Level [40], refers to how carefully an institution vets a user's identity claim (for example, at the time of hiring). IAL1 represents practically no control whereas IAL3 has tight controls such as requiring a positive ID presented during physical presence at the Identity Provider when the initial assignment of new login credentials is handled. AAL is Authentication Assurance Level and refers to how carefully controlled the provider's login process is [41]. AAL1 can be as simple as just username and 30asswordd entry. AAL3 requires multifactor authentication, one of which must be hardware based, and it requires some means for the user to tell that the login handler is genuine and not being spoofed. FAL is Federation Assurance Level and describes the thoroughness of the networking protocols [42], with FAL1 being a simple encryption of the authorization response up to FAL3 with a multistep protocol involving certificates and encryption to control what information can be known by friend or foe at each step.

Supply Chain Attacks and Trust

Modern software development practices rely heavily on third party code from sources that are poorly understood. From a productivity standpoint this is wonderful, but it represents a colossal security exposure. It's not a new threat at all, though. Mitigation comes down to knowing your exposures, your sources, and the level of trust you can assign those parties [43].

To be clear, manipulating the contents of software and getting a target to run that code is not restricted to Open Source software, as is so often assumed. Old and possibly apocryphal tales of a prankster mailing a fake update tape to the Computing Center were told in the past tense in the 80s. Supposedly this official-

looking tape contained a malicious patch that would let the graduate students play Adventure without billing records showing up. Even if the stories were tall tales, the idea was widespread.

It is important that the sources of external software are known. This has two components: deciding on your level of trust in each supplier and verifying that the software they wrote is actually what you're running. The first case is by definition a value judgment and is resistant to meaningful quantification. It is possible to know the authors of a very specialized piece of software with a small community around it. In that case, reputation matters. It is harder to know anything about the developers in large projects. In that case, proxy measures are all we have. Is the software part of a major Linux distribution? That's not enough on its own. Has the code been through a formal, third-party review? Again, that is not foolproof at all, but it is a good sign.

The second part of knowing your supplier is knowing that the software they produced is what you are actually running. It is not unheard of at all for a third party to infiltrate the development process and change something. Many software distribution methods use public key encryption to "sign" their code[44]. When installing it from a repository (repo) that does this, the code is downloaded and the signature is verified before installation. Some repositories do this well (Ubuntu uses PKI certificates for their own supported packages) and some do not do this at all (CPAN, the archive for Perl packages, has limited support [45]).

One of those most important mitigations against supply chain attacks is simply being able to respond quickly enough when problems are found. To this end, it is critically important to have an inventory of software packages running across your cyberinfrastructure. In the case of the recent Log4J vulnerability [46], fixing the problem was very simple - install the new version of the package and clean up any damage from the attacks themselves. A significant challenge was trying to determine what servers were affected and where (potentially multiple!) copies of Log4J were installed.

The software supply chain is a target-rich environment. The work to compromise a repository can pay off handsomely in the sheer depth and breadth of a possible attack. One encouraging sign is the number of repos that are considering or planning to adopt both repository signing ("this package really was downloaded from the repository you expected") and author signing ("this editor really was written by Richard Stallman").

Self-Sovereign Identity

This section on Self-Sovereign Identity (SSI) is derived from the SSI Presentation to the CI Compass Identity Management Working Group on March 14th, 2022, and available on the CI Compass web site [47].

For a user of an information system that needs to keep up with more than one user, that user has to have an online identity and that identity is the key to their digital existence. We are known professionally by our identity in that context. UNC's assertion that I'm scotte@unc.edu is basic to my ability to do work. Not surprisingly, we have a lot of identities. For many of my personal matters, I'm Erik.L.Scott@gmail.com. In my private life, I'm "Erik" or "Dad" or any number of other monikers that are situationally appropriate. Managing those identities and being able to control them (or not) determines a large degree of my agency.

Self-Sovereign Identity is the control of a digital identity by the human that uses it without any third party being able to intervene.

Full control of our identities is an important goal. As an example, see Limor Fried's case. She is known professionally and online as "Ladyada". Google's online services recognize her by that identity. Internal algorithms combined with a data slurping mission at Google decided that the use of a nickname was inadequate, resulting in her account being suspended. This profoundly affected her ability to do business. The case was resolved largely because she's "Internet Famous" and was able to agitate for her case via the technical press and a few other platforms.

Even in the absence of hostile actions, there are other reasons to want full control of our identities and to be able to assert them in a reliable way. One example is privacy. If I use Google's authentication service to access another party's assets (say, using my Google ID to update my Wordpress blog) then I have to wonder if The Big G has recorded this and will use it for some unspecified purpose. I'm also vulnerable to the reliability of a third party. There have been, for instance, occasional outages of Microsoft's Office 365 and their Azure Active Directory service. Microsoft is far from the only provider who has had temporary failures of their IdM infrastructure, and while any of these third parties may have excellent operational histories, I have no control over how they prioritize recovery efforts. What I want is a system that is highly resistant to damage and has no single points of failure.

Interest in SSI is not confined to the paranoid fringe, but rather is gaining momentum across both government and commercial domains. In the government world, identifying citizens is a fundamental task. The European Union is in the lead here through their GAIA-X project [48], which is part of their European Sovereignty strategy. Internet users in general, including of course the EU, have a substantial dependency on IdM from the "FAANG" companies: Facebook, Apple, Amazon, Netflix, and Google. These are all US companies with interests sometimes in direct opposition to EU regulations.

The EU does not represent the only public sector interest, to be sure. Interest in this direction in the US has been seen in the Federal Government, a couple of states, and even some local jurisdictions. Some of this has been driven by the failures (or, more likely, public awareness of the failures [49]) of third party IdM contractors. A recent example is the negative press surrounding the IRS's use of "ID.me" for controlling the ability to access their public-facing systems. ID.me data gathering program was offensively extreme.

SSI interest is not confined to governments. There is notable commercial interest from Finance, Consulting, and Federal Contractors. This last group is not surprising; they have a strong interest in understanding their customer. Consulting groups count financial customers among their largest clients. Finance has an interest in SSI for both privacy reasons and for compliance with "Know Your Customer" laws related to money laundering and tax evasion. Banks would like to have a way of managing customer identities that is both safer than trusting a random Google account and less expensive than maintaining one of the most tempting cybersecurity targets imaginable… their authentication data.

All statements of identity will ultimately be determined by the assertions of others. If I act purely on my own, I could declare that I'm "L. Bob Rife" and, absent the ability for anyone else to protest, the world would have to assume I'm right. This problem is not unique to SSI. There was a chain of attestations when I was born. My mother asserted that my name was Erik and the nurses asserted that she was my mother. Both parties signed my birth certificate. My identity was therefore asserted by two parties. SSI has a similar problem, and will require some means to verify and make assertions about a particular person's identity. This will likely involve a catalog of publicly-visible assertions about me from a variety of sources with varying degrees of trustworthiness. At some point, verification of my identity when I log in somewhere will depend on a model based on the number and quality of these assertions. Similarly, revocation of identity is also a

potential problem. A case where this could be necessary is when identity fraud is detected and the original attestations need to be revoked. A reasonable solution is for the original parties to issue revocations, not unlike a journal article retraction today.

A reliable, publicly-visible, and secure infrastructure for storing SSI attestations and validating their authenticity requires some form of distributed, tamper-proof “ledger.” One solution to this lies in the hottest of buzzwords: Blockchain. Blockchain fits the requirements, modulo sufficient performance and scalability enhancements. The distributed nature makes it highly reliable and resistant to fairly sophisticated attacks. The use of cryptographically signed records with strong tamper detection makes it trustworthy. While Blockchain is usually associated with cryptocurrencies, it’s ultimately a distributed ledger. Any such system could be used instead, and some offer more versatility, such as storing complex provenances in a more discoverable way.

In the short term, Major Facilities should continue to use existing Federated IdM strategies. SSI is still largely a research topic (though it’s being used operationally for a few government tasks in Belgium, for instance). Nonetheless, Major Facilities often have at least a few EU users and will sometimes have users not affiliated with an InCommon institution. SSI might well become a feasible alternative to the FAANGs in situations like this and it bears watching. In the meantime, an IdM provider such as ORCID represents a good alternative.

References and Resources

[1] Multi-factor Authentication: https://en.wikipedia.org/wiki/Multi-factor_authentication

[2] InCommon: https://incommon.org/

[3] AARC Blueprint Architecture: https://aarc-project.eu/architecture/

[4] A good overview of the SAML protocol: https://en.wikipedia.org/wiki/Security_Assertion_Markup_Language

[5] Another good explanation of SAML: https://auth0.com/blog/how-saml-authentication-works/

[6] Duo has a really good introduction to SAML, and their style is even cheekier than The Cookbook: https://duo.com/blog/the-beer-drinkers-guide-to-saml

[7] The official word on OIDC: https://openid.net/connect/

[8] Another good overview of OIDC: https://auth0.com/docs/protocols/openid-connect-protocol

[9] OIDC Protocol in fine detail: https://openid.net/specs/openid-connect-core-1_0.html

[10] SAML vs. OIDC, head-to-head: https://auth0.com/intro-to-iam/saml-vs-openid-connect-oidc/

[11] A comprehensive collection of links to libraries for implementing OIDC: https://openid.net/developers/libraries/

[12] OAuth2 RFC ("internet standards document"): https://tools.ietf.org/html/rfc6749

[13] It used to be called "Simple Cloud Identity Management", hence the url: http://www.simplecloud.info/

[14] RFC 7521, OIDC Assertions: https://datatracker.ietf.org/doc/html/rfc7521

[15] What the web site shows depends on whether or not you're logged into it. If you get confused, and I frequently do, use an "Incognito" window and try again with a clean slate of cookies: https://www.cilogon.org/home

[16] EduGain, a worldwide identity federation: https://edugain.org/

[17] Globus: https://www.globus.org/

[18] JWT Claims Defined by IANA https://www.iana.org/assignments/jwt/jwt.xhtml

[19] Wikipedia has a good, quick introduction. Especially useful for the "References" section at the end: https://en.wikipedia.org/wiki/X.509

[20] LDAP information galore: https://ldap.com/

[21] The OpenLDAP implementation: https://www.openldap.org/

[22] Long and detailed introduction to AD, from the source: https://technet.microsoft.com/en-us/library/bb727030.aspx

[23] Samba: https://www.samba.org/

[24] Internet2: https://internet2.edu

[25] Containerized, for your convenience: https://incommon.org/trusted-access/

[26] https://incommon.org/software/shibboleth/

[27] https://incommon.org/software/comanage/

[28] https://incommon.org/software/grouper/

[29] Jim Basney, Terry Fleury, and Jeff Gaynor, "CILogon: A Federated X.509 Certification Authority for CyberInfrastructure Logon," Concurrency and Computation: Practice and Experience, Volume 26, Issue 13, pages 2225-2239, September 2014. https://doi.org/10.1002/cpe.3265

[30] Satosa introduction: https://daasi.de/en/satosa-a-modular-proxy/

[31] Satosa Introduction II: https://aarc-project.eu/training/what-is-satosa/

[32] Satosa Training: https://aarc-project.eu/wp-content/uploads/2019/03/SaToSa_Training.pdf

[33] Microsoft Active Directory Federation Services: https://docs.microsoft.com/en-us/windows-server/identity/active-directory-federation-services

[34] Office 365 is one of the options, as is “Free”: https://azure.microsoft.com/en-us/pricing/details/active-directory/

[35] Auth0: https://auth0.com/

[36] Microsoft’s Azure Active Directory: https://azure.microsoft.com/en-us/services/active-directory/

[37] PingIdentity: https://www.pingidentity.com/en.html

[38] SimpleSAMLphp: https://simplesamlphp.org/

[39] General document collection for NIST 800-63: https://www.nist.gov/identity-access-management/nist-special-publication-800-63-digital-identity-guidelines

[40] The conformance criteria checklists are especially convenient, such as this one for IAL: https://www.nist.gov/document/conformance-criteria-sp-800-63a-enrollment-and-identity-proofing

[41] AAL checklist: https://pages.nist.gov/800-63-3-Implementation-Resources/63B/AAL/

[42] FAL checklist: https://pages.nist.gov/800-63-3-Implementation-Resources/63C/SecurityParameters/

[43] Section 3.5 (Software Supply Chain) from the Trusted CI Guide to Security Scientific Software, https://doi.org/10.5281/zenodo.5777646

[44] This example is for Ubuntu, but essentially all Linux distros work similarly. Selected for its ease of understanding: https://askubuntu.com/questions/131397/what-is-a-repository-key-under-ubuntu-and-how-do-they-work

[45] Not only that, but the default is to not check the signatures. https://blog.hackeriet.no/cpan-signature-verification-vulnerabilities/

[46] Log4J is very widely used, hence the large scale of the problem. For a gentle introduction, see https://www.cygenta.co.uk/post/log4shell-in-simple-terms . For the gory details, look at https://blog.shiftleft.io/log4shell-jndi-injection-via-attackable-log4j-6bfea2b4896e

[47] https://ci-compass.org/

[48] An overview of GAIA-X: https://www.data-infrastructure.eu/GAIAX/Navigation/EN/Home/home.html

[49] The Register, in their refreshingly blunt style, took advantage of a soft target: https://www.theregister.com/2022/02/07/irs_facial_plan_scrapped/

Glossary

AARC: Authentication and Authorization for Research and Collaboration. Part of the EU Horizon program. Publisher of the AARC Blueprint.

AARC Blueprint: A high-level list of requirements that any IdM design must meet for federation within EU institutions and organizations.

Active Directory (AD): Microsoft's means for looking up information on users and systems – essentially "Microsoft's LDAP Competitor". AD can in fact provide services via LDAP as well. Development pace has slowed or nearly stopped in favor of their Azure Active Directory, which while largely compatible is not just "AD running on a cloud instance".

ADFS (Active Directory Federation Services): Microsoft's federation layer for Active Directory (not Azure AD). Can be made to interoperate with InCommon protocols albeit with substantial effort. Appears at or near end-of-life status already.

Azure Active Directory (Azure AD or AzureAD): Microsoft's new IdM solution, hosted on their Azure Cloud, and largely or completely supplanting conventional, on-premises AD.

CILogon: see CILogon2

CILogon2: Software-as-a-Service and, much more importantly, agreements between effectively thousands of institutions that allows a service provider to use the Single Sign On system of another institution with the providers' own systems. Main access method is OIDC. Example: my Duke University credentials can be used through Duke's Shibboleth to grant access to UNC's social science data enclave.

CoManage: InCommon software for managing group assignments of users ("faculty" vs "student", for instance). Several automation tools for common tasks. Some help for cases where a single human has multiple (computer) identities.

Distributed Computing: an architectural style where tasks are divided among many separate computers, typically with a given type of task always running on a specific machine.

Django: Large, richly complex framework for developing web applications in Python. Has its own simplistic IdM that can be replaced transparently to use modern IdM. See especially mozilla-django-oidc, below.

Duo Mobile: A cellphone application for Multi-Factor Authentication.

Federation: The act of different IdM staff and systems, typically at different institutions, trusting each other sufficiently to allow one to Authenticate users on behalf of the other. Example: my Duke University username and password is allowed to log in to UNC's private enclave for social sciences.

Google Authenticator: A cellphone application for Multi-Factor Authentication.

Grouper: InCommon software for managing large, complicated group memberships. Handles large rulesets and can flag logical inconsistencies.

Identity Assurance: Not a technology per se but instead the process of having operational standards and controls in place so institutions can mutually trust each other's IdM assertions. Example: all InCommon members agree to require picture ID's to reset passwords.

Identity Management: Keeping track of who a user is, whether they probably are who they say they are (perhaps with passwords, two factor authentication, or biometrics), and what they're allowed to do. These are the roots of Authentication (AuthN) and Authorization (AuthZ).

IdM: see Identity Management

InCommon: Originally Internet2's standards body for federated IdM, now an independent non-profit organization in its own right.

Internet2: A non-profit consortium of research universities and research organizations that operate a high performance network linking them and independent of commercial internet providers. Defines standards for, among other things, federated identity management.

JSON Web Tokens (JWT): identity information encoded as JSON. Contains a digital signature to verify the contents and to detect tampering. Typically has a defined lifetime and can be used for many requests until it finally expires. Commonly seen in conjunction with OAuth2 and OIDC.

LastPass Application Client: A cellphone application for Multi-Factor Authentication.

Lightweight Directory Access Protocol (LDAP): A venerable technology supporting the querying of user data. Commonly used for verifying login credentials and email directory lookups throughout an institution.

Mozilla-django-oidc: add-on module for Django. Makes using an OIDC identity provider look like using Django's default, built-in IdM.

Multi-factor Authentication: Determining that someone is who they say they are requires two or more actions. Typically this means "typing a password" and "having some widget with you like your cellphone so the IdM system at least determines you're more than likely who you say you are".

OAuth: see OAuth2

OAuth2: IdM Protocol over HTTP. Makes explicit distinctions between Resource Owners and Resource Severs, and between indications of authorization versus separate grants of rights for each protected access.

OpenID Connect (OIDC): IdM Protocol based on JSON Web Tokens (JWT) which are human-readable JSON describing names, email addresses, and similar identity information. Built on top of OAuth2.

PingIdentity: Commercial software and cloud vendor of IdM tools. Can serve SAML, OAuth, and OIDC.

Samba: Software package mainly used for sharing directories and printers to Windows instances from Linux and Unix servers. Was once able to handle Windows authentication requests, obsolete for this purpose now.

SAML: Security Assertion Markup Language. An XML format for exchanging identity information, typically between Identity Providers and Service Providers.

Satosa: “SAML to SAML” proxy. Can act as a protocol converter to bridge disparate systems, and can edit SAML messages to add or remove information.

Self-Sovereign Identity: An IdM scheme where individuals can attest to their identity without requiring a centralized authority. Often relies on tamper documents, often blockchain. Significant European interest in response to US Companies dominating the public space here.

Shibboleth: InCommon protocol and software for Single Sign On (SSO).

Supply Chain Attack: Slipping software with security holes into components and libraries used by intended victims. Example: random strangers sticking backdoors into NPM packages; bored students sending fake software updates to unsuspecting users.

System for Cross-domain Identity Management (SCIM): IdM communication protocol, using JSON, for exchanging either single records or bulk quantities of data between IdM servers. Among other uses, often seen exchanging data between institutional IdM servers and Software-as-a-Service (SaaS) providers.

Yubico: Vendor; sells hardware devices, historically with USB interfaces, that serve as the “something you have” component of Multi-Factor Authentication. New products offer Near Field Communication as an alternative to USB.

Acknowledgements

The authors would like to thank several individuals who have made this guide possible: John Haverlack, Chris Bontempi, Robert Casey, Doug Ertz, and the other members of the CI Compass Identity Management (IdM) Working Group, who shared their perspectives on Major Facilities and guided us in selecting IdM topics of broad interest. We would also like to thank Jim Basney for his advice on IdM topics, including CILogon and CILogon2. Thanks is also extended to other CI Compass and Trusted CI team members for their help in writing this document.