Setup a Single Sign On SAML Test Environment with Docker and NodeJS

Jeffry Houser Follow
Jul 11, 2019 · 17 min read

I'm Jeffry Houser, a developer from the Polaris team in the content engineering group at Disney Streaming Services. Polaris was named after Magneto's daughter from the X-men, and we builds internal tools that allow editors to create magic for our customers in the video services we power.

When working on one of our tools, we needed to Integrate with a single sign on system that uses SAML for authentication. Setting up a local environment for testing SAML was not a trivial task. A lot of articles we found recommended using Feide OpenIdP as a test provider, however that shut down years ago. Additionally, many of the code samples used outdated libraries, leaving some gaps in our knowledge. It took some trial and error to piece together a working solution and I'm going to teach you how we did it.

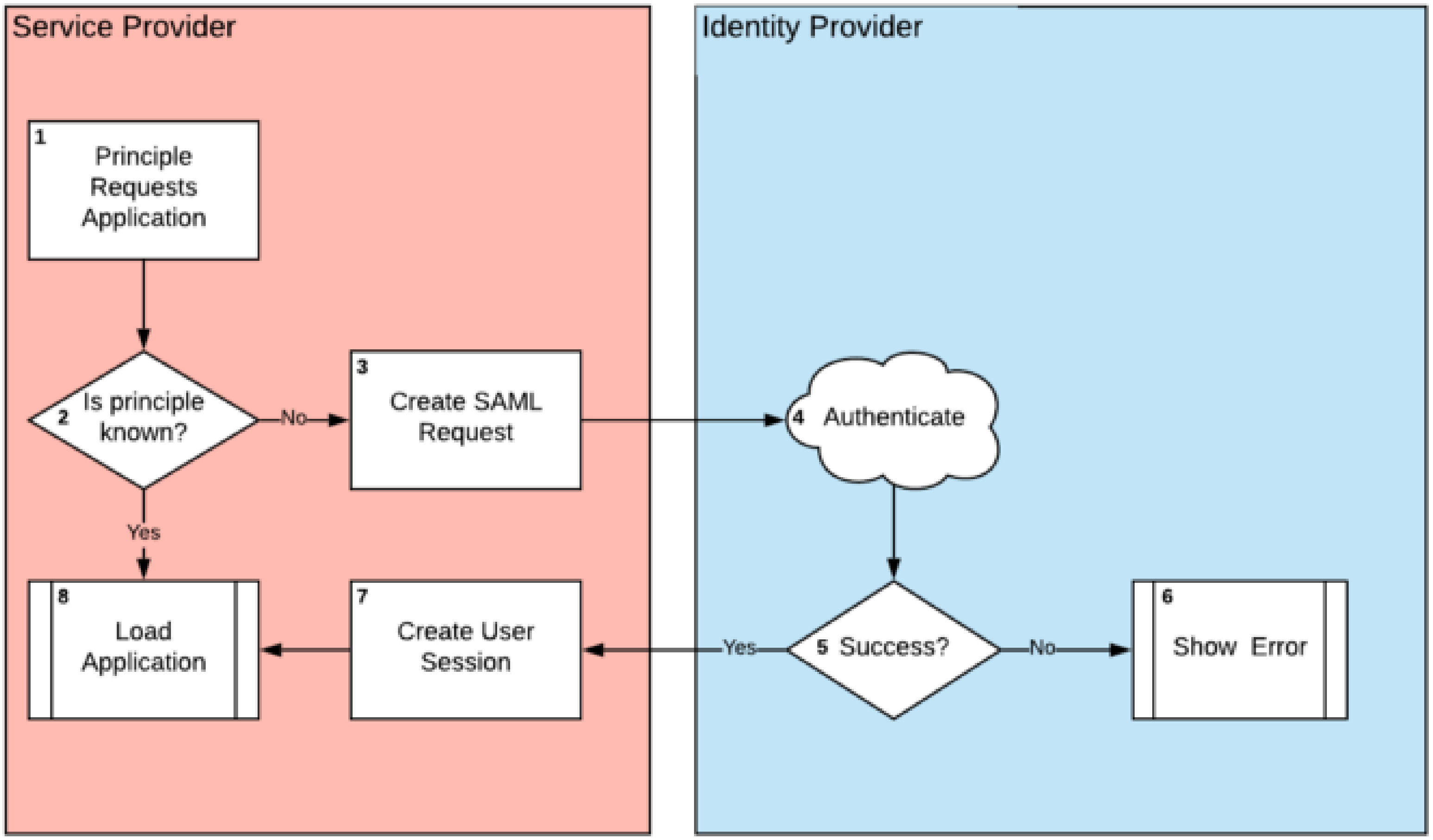
The first time I was exposed to it; SAML was difficult for me to get my head around. As such, I'm going to start out with some definitions that will help you understand the pieces of a SAML application.

- **Single Sign On (SSO)**: Any system that allows authentication code and login data to be shared across multiple applications.
- **Security Assertion Markup Language (SAML)**: A framework, and XML schema, for implementing Single Sign On.
- **Principal**: The user who is attempting to gain access to our application.
- **Assertions**: Data about the principal which are included as part of the SAML response. Samples of this might be the user's name, or other permission data.
- **Service Provider (SP)**: This is the application, or system, that the user is attempting to access. We will build a simple SP as part of this article.
- **Identity Provider (IdP)**: This is a remote application, or system, that authenticates the user and returns data back to the service provider. We're not going to build an IdP from scratch, but I'll show you how to set up and use a pre-built one.
- **Globally Unique Identifier**: A value that the IdP will use to identify an SP.

even more important and I'll go over that next.

Review the Application Flow

This is a common flow for a SAML application:



1. The Principal — AKA User — tries to access your Service Provider — AKA your application.
2. The Service Provider checks to see if it knows the Principal. In a browser-based app, this session information would probably be stored as a cookie, but a desktop or application server may store that information in memory. If the user is known, we can load the app normally, so move onto step 8. If the user is not known, jump to step three to start the authentication process.
3. If no user is known, the SP creates a SAML Request and sends that request to the IdP. This request will contain the Globally Unique Identifier so that the IdP knows which application the principal requested access to.
4. Now the IdP handles the request. It will authenticate the user. It may do this based on an existing session from a previous sign in, or it may have the user login anew.
5. Did the IdP successfully collect user details on the Principal? If so, go to step 7, the success set. Otherwise go to step 6 the failure step.
6. If the Principal was not able to login, the IdP will handle authentication errors and the SP will know nothing about the failure.

7. If the IdP successfully logged the user in, it will create a SAML response packet including assertions about the user, and send the info back to the SP's callback URL. The SP will use that data to create a user session.

8. If the Principal authenticated properly, then load up the app and let them in.

The rest of this article will focus on steps 3 through 7.

Install Prerequisites

You'll want to install a prerequisites before we start jumping into the code:

- Docker: Docker is a container platform that lets us easily create virtual machines with predefined code. We're going to use it to easily create our own Identity Provider.

- NodeJS: We are going to write our Service Provider from scratch using a NodeJS and some common plugins.

- OpenSSL: OpenSSL can be used to create public and private key certificates for. Certs like these are often used for SSL on web sites, but we're going to use them to encrypt and decrypt the packets we're sharing between our SP and IdP.

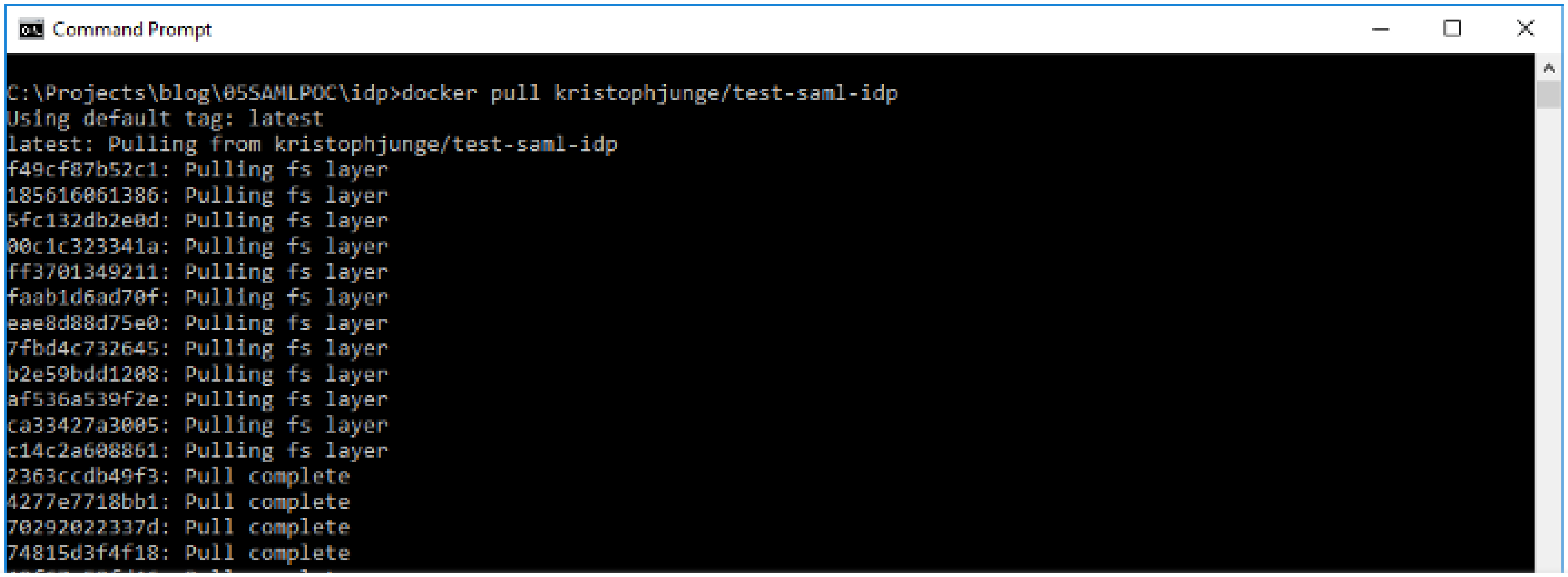
The install instructions at the respective sites will give better setup instructions than anything I could provide here.

Creating an identity provider is hard and complicated, so we're going to use an application that is easily configurable and will run it in a docker container. We're going to use the SimpleSamPHP IdP application and run it in an existing docker container.

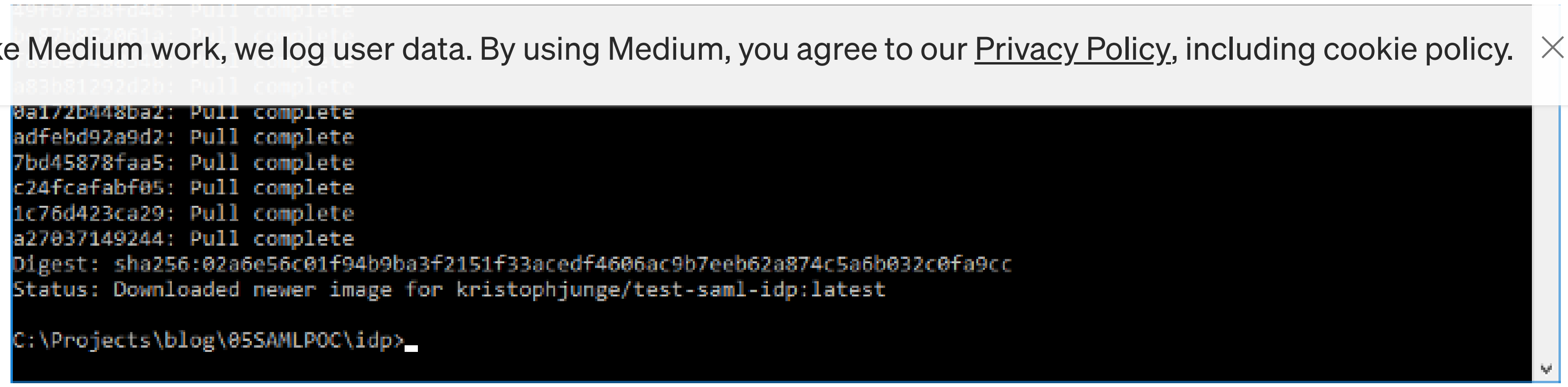
The container is already loaded into the Docker hub, so we can download that without needing to build it from the source. Run this docker command at your command line:

```
docker pull kristophjunge/test-saml-idp
```

You'll see something like this:



```
C:\Projects\blog\05SAMLPOC\idp>docker pull kristophjunge/test-saml-idp
Using default tag: latest
latest: Pulling from kristophjunge/test-saml-idp
f49cf87b52c1: Pulling fs layer
185616061386: Pulling fs layer
5fc132db2e0d: Pulling fs layer
00c1c323341a: Pulling fs layer
ff3701349211: Pulling fs layer
faab1d6ad70f: Pulling fs layer
eae8d88d75e0: Pulling fs layer
7fbd4c732645: Pulling fs layer
b2e59bdd1208: Pulling fs layer
af536a539f2e: Pulling fs layer
ca33427a3005: Pulling fs layer
c14c2a608861: Pulling fs layer
2363ccdb49f3: Pull complete
4277e7718bb1: Pull complete
70292022337d: Pull complete
74815d3f4f18: Pull complete
```



```
0a172b448ba2: Pull complete
adfebd92a9d2: Pull complete
7bd45878faa5: Pull complete
c24fcafabf05: Pull complete
1c76d423ca29: Pull complete
a27037149244: Pull complete
Digest: sha256:02a6e56c01f94b9ba3f2151f33acedf4606ac9b7eeb62a874c5a6b032c0fa9cc
Status: Downloaded newer image for kristophjunge/test-saml-idp:latest

C:\Projects\blog\05SAMLPOC\idp>
```

Now you should be able to run the docker image:

```
docker run — name=testsamlidp -p 8080:8080 -p 8443:8443
-e SIMPLESAMLPHP_SP_ENTITY_ID= saml-poc
-e
SIMPLESAMLPHP_SP_ASSERTION_CONSUMER_SERVICE=http://localhost:4300/log
in/callback
-d kristophjunge/test-saml-idp
```

There is a lot going on in this command, and it can be confusing if you are not familiar with Docker. Let’s look at each part of the command:

- **docker run**: This tells docker to run a new container
- **— name=testsamlidp**: This tells Docker that the name of our local container will be testsamlidp.

- **-p 8080:8080 -p 8443:8443**: This is a port mapping. Docker listens to the external port 8080 and maps it to the internal port 8080. Additionally, it listens to port 8443 will map to port 8443 in the docker container. The two ports are for http and https traffic into our IdP.

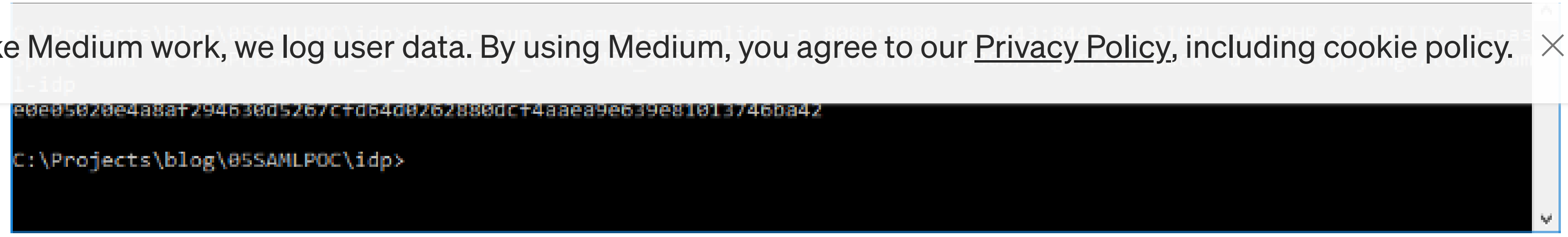
- **-e SIMPLESAMLPHP_SP_ENTITY_ID=saml-poc**: This passes in an argument to our docker container. This defines the Globally Unique Identifier for the service provider. We'll use this value in our SP code later

- **-e SIMPLESAMLPHP_SP_ASSERTION_CONSUMER_SERVICE=http://localhost:4300/login/callback**: This is another argument we're passing into the docker container. We're telling it where to redirect to after a successful login. When we build out the Service Provider, it will be on port 4300 at our localhost.

- **-d:** The d argument tells us to run the container in the background, and print out the ID.

- **kristophjunge/test-saml-idp**: This tells docker which image to use for our container.

Run the command and you'll see something like this:

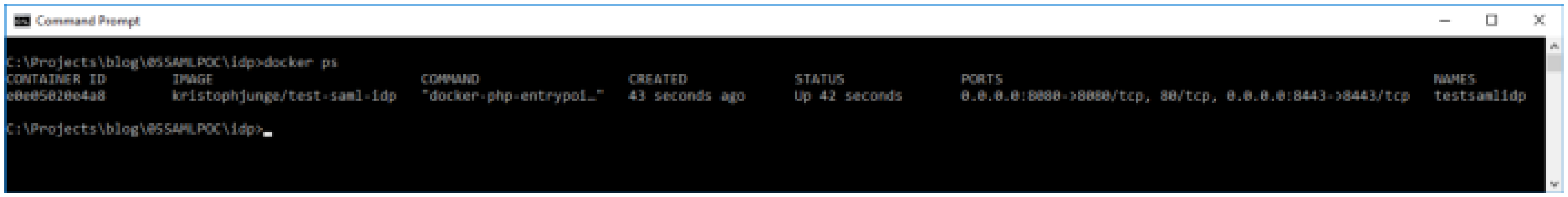


```
e0e05020e4a8af294630d5267cfd64d0262880dcf4aaea9e639e81013746ba42

C:\Projects\blog\05SAMLPOC\idp>
```

You can run this command

```
docker ps
```

to make sure that the docker image is running:

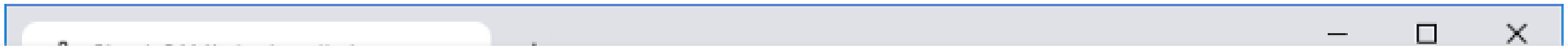


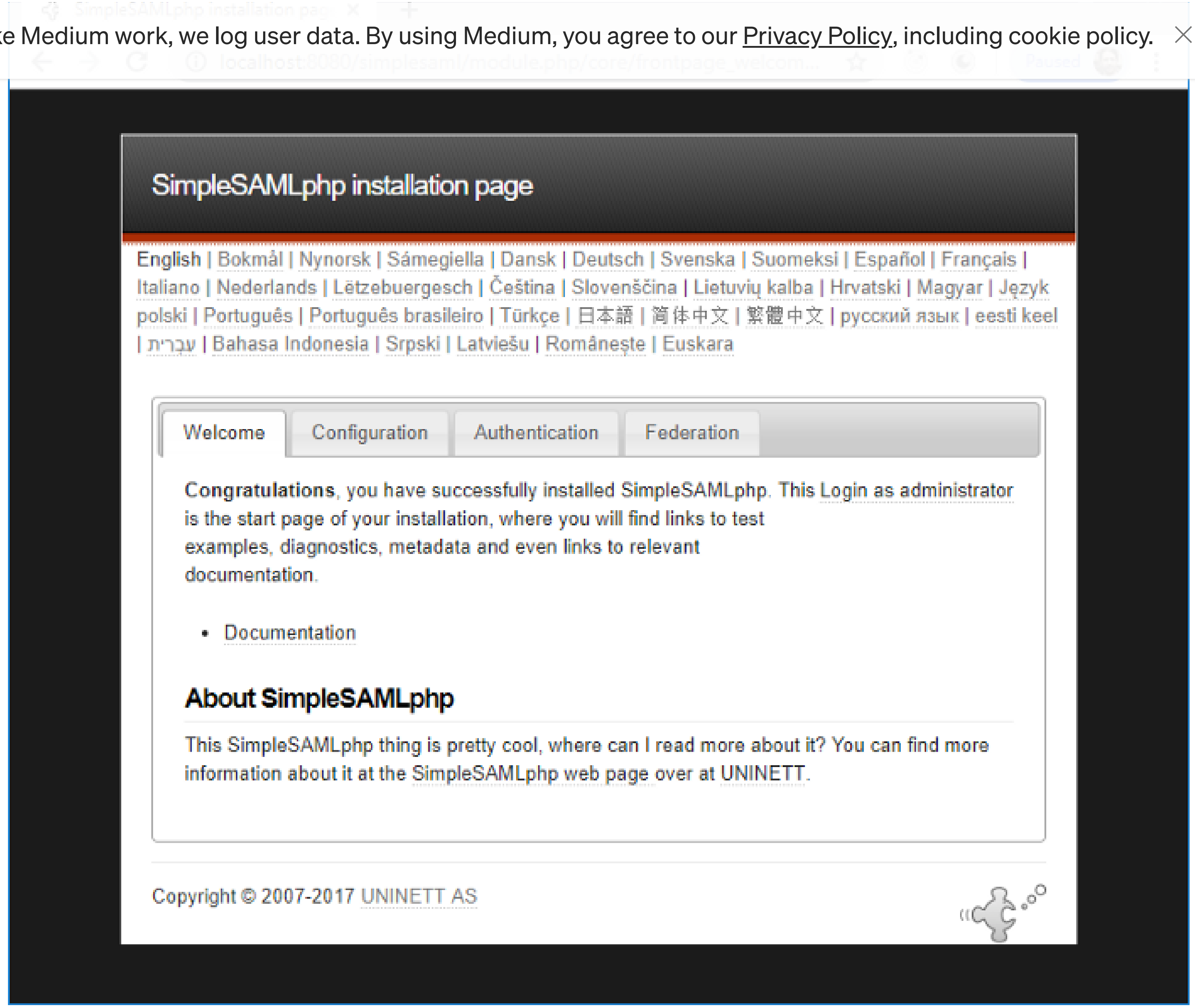
```
C:\Projects\blog\05SAMLPOC\idp>docker ps
CONTAINER ID        IMAGE                        COMMAND                  CREATED             STATUS              PORTS                                                        NAMES
e0e05020e4a8        kristophjunge/test-saml-idp   "docker-php-entrypoi…"   43 seconds ago      Up 42 seconds       0.0.0.0:8080->8080/tcp, 80/tcp, 0.0.0.0:8443->8443/tcp      testsamlidp

C:\Projects\blog\05SAMLPOC\idp>
```

Try to load the SAML IdP provider in your browser by going to this URL:

http://localhost:8080/simplesaml

You should see something like this:




SimpleSAMLphp installation page
English | Bokmål | Nynorsk | Sámegiella | Dansk | Deutsch | Svenska | Suomeksi | Español | Français | Italiano | Nederlands | Lëtzebuergesch | Čeština | Slovenščina | Lietuvių kalba | Hrvatski | Magyar | Język polski | Português | Português brasileiro | Türkçe | 日本語 | 简体中文 | 繁體中文 | русский язык | eesti keel | עִבְרִית | Bahasa Indonesia | Srpski | Latviešu | Românește | Euskara
Welcome
Configuration
Authentication
Federation
Login as administrator
Documentation

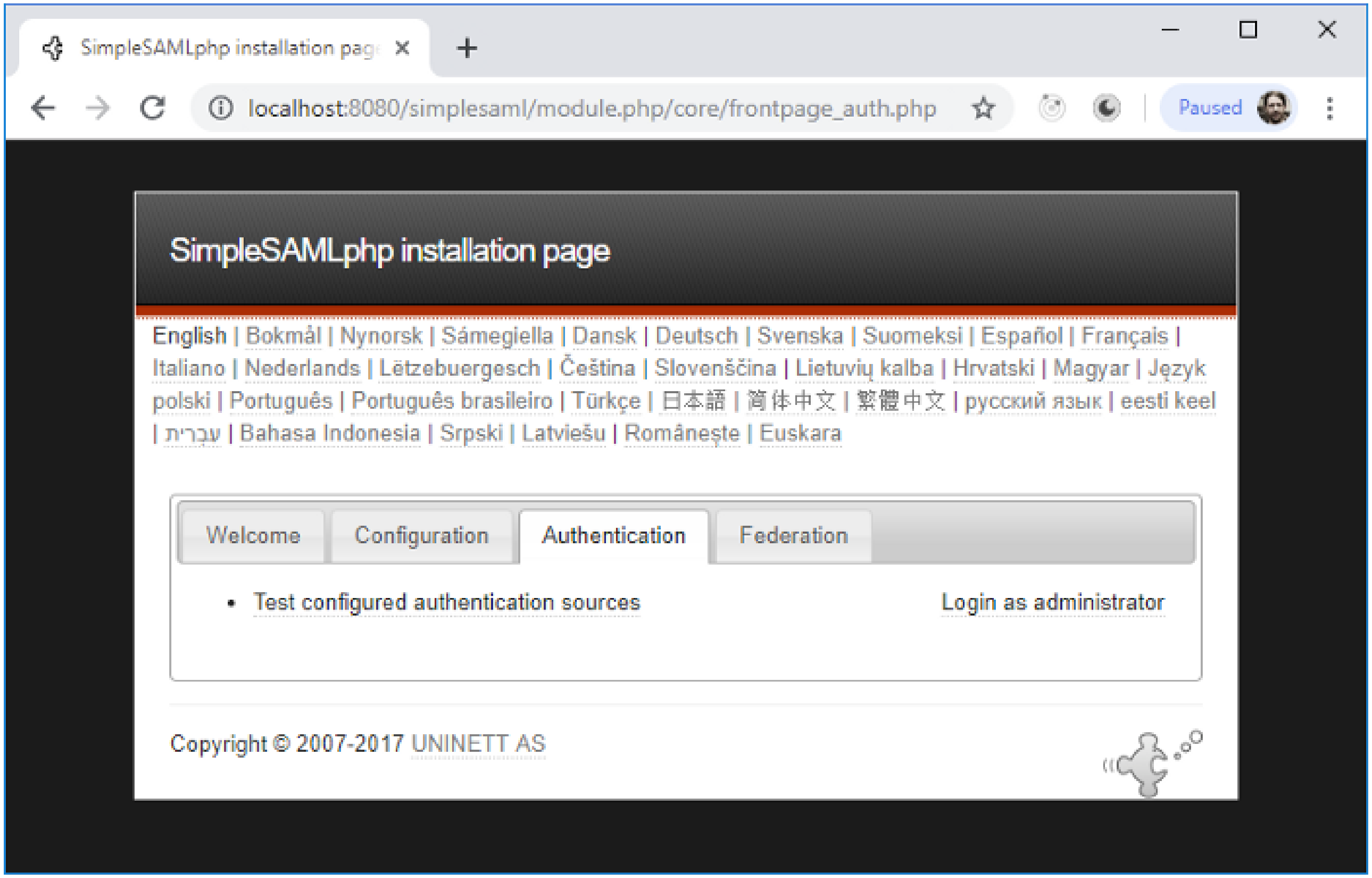
Congratulations, you have successfully installed SimpleSAMLphp. This is the start page of your installation, where you will find links to test examples, diagnostics, metadata and even links to relevant documentation.

- Documentation

About SimpleSAMLphp

This SimpleSAMLphp thing is pretty cool, where can I read more about it? You can find more information about it at the SimpleSAMLphp web page over at UNINETT.

Copyright © 2007-2017 UNINETT AS


SimpleSAMLphp installation page
localhost:8080/simplesaml/module.php/core/frontpage_auth.php
Paused

SimpleSAMLphp installation page

English | Bokmål | Nynorsk | Sámegiella | Dansk | Deutsch | Svenska | Suomeksi | Español | Français | Italiano | Nederlands | Lëtzebuergesch | Čeština | Slovenščina | Lietuvių kalba | Hrvatski | Magyar | Język polski | Português | Português brasileiro | Türkçe | 日本語 | 简体中文 | 繁體中文 | русский язык | eesti keel | עִבְרִית | Bahasa Indonesia | Srpski | Latviešu | Românește | Euskara

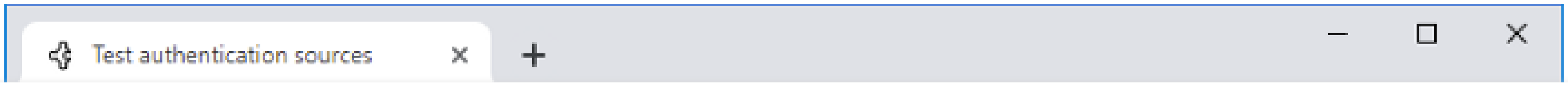
Welcome | Configuration | Authentication | Federation

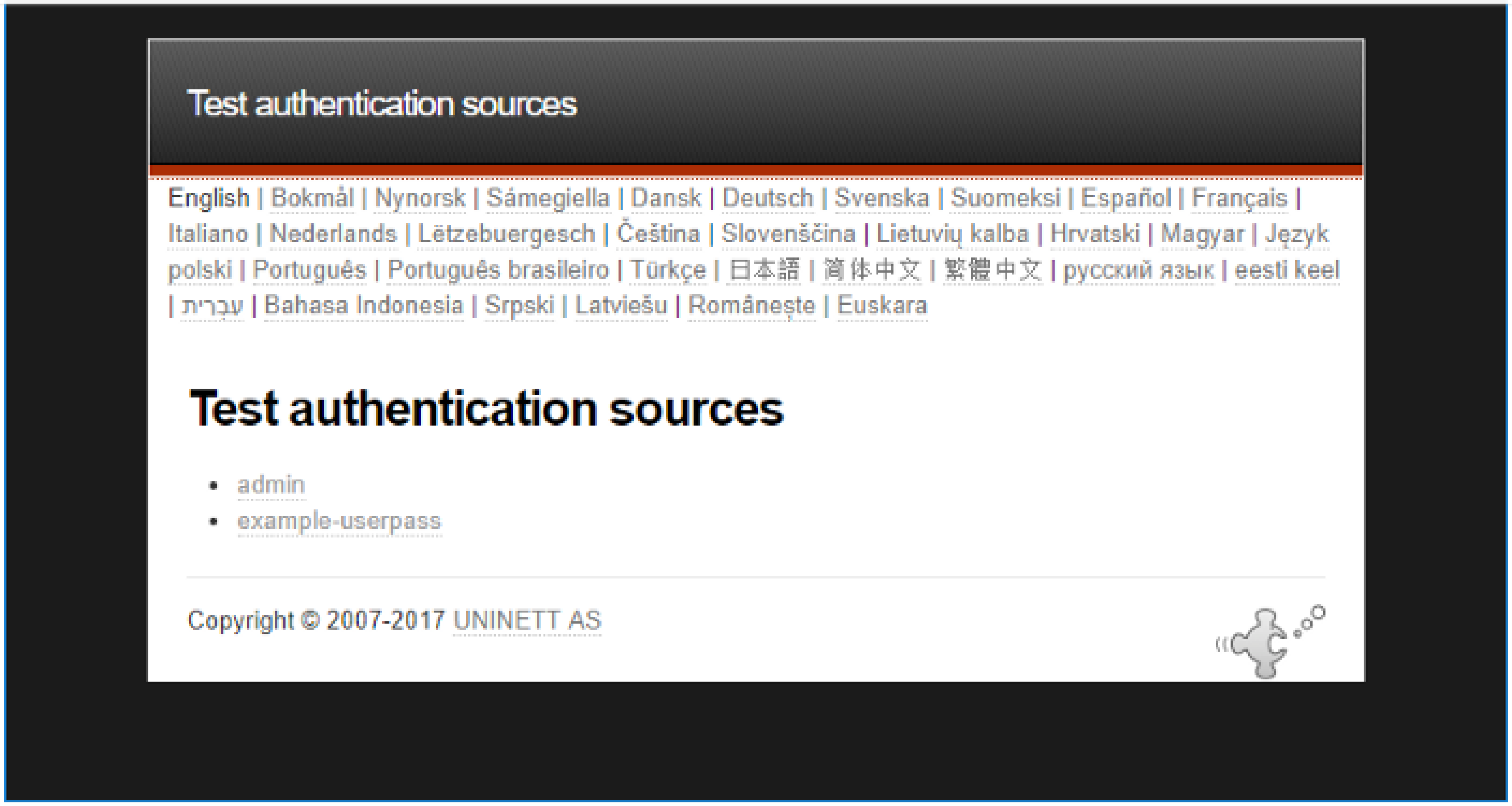
- Test configured authentication sources

Login as administrator

Copyright © 2007-2017 UNINETT AS

Then click on Test configured authentication sources which will bring you to





Test authentication sources

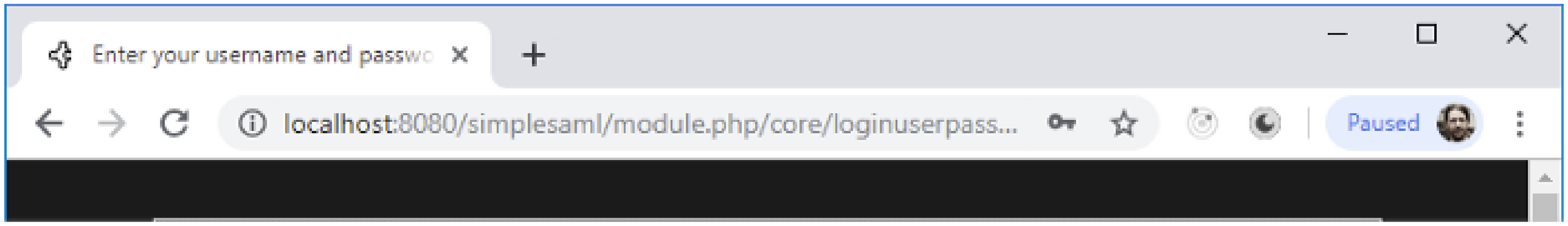
English | Bokmål | Nynorsk | Sámegiella | Dansk | Deutsch | Svenska | Suomeksi | Español | Français | Italiano | Nederlands | Lëtzebuergesch | Čeština | Slovenščina | Lietuvių kalba | Hrvatski | Magyar | Język polski | Português | Português brasileiro | Türkçe | 日本語 | 简体中文 | 繁體中文 | русский язык | eesti keel | עִבְרִית | Bahasa Indonesia | Srpski | Latviešu | Românește | Euskara

Test authentication sources

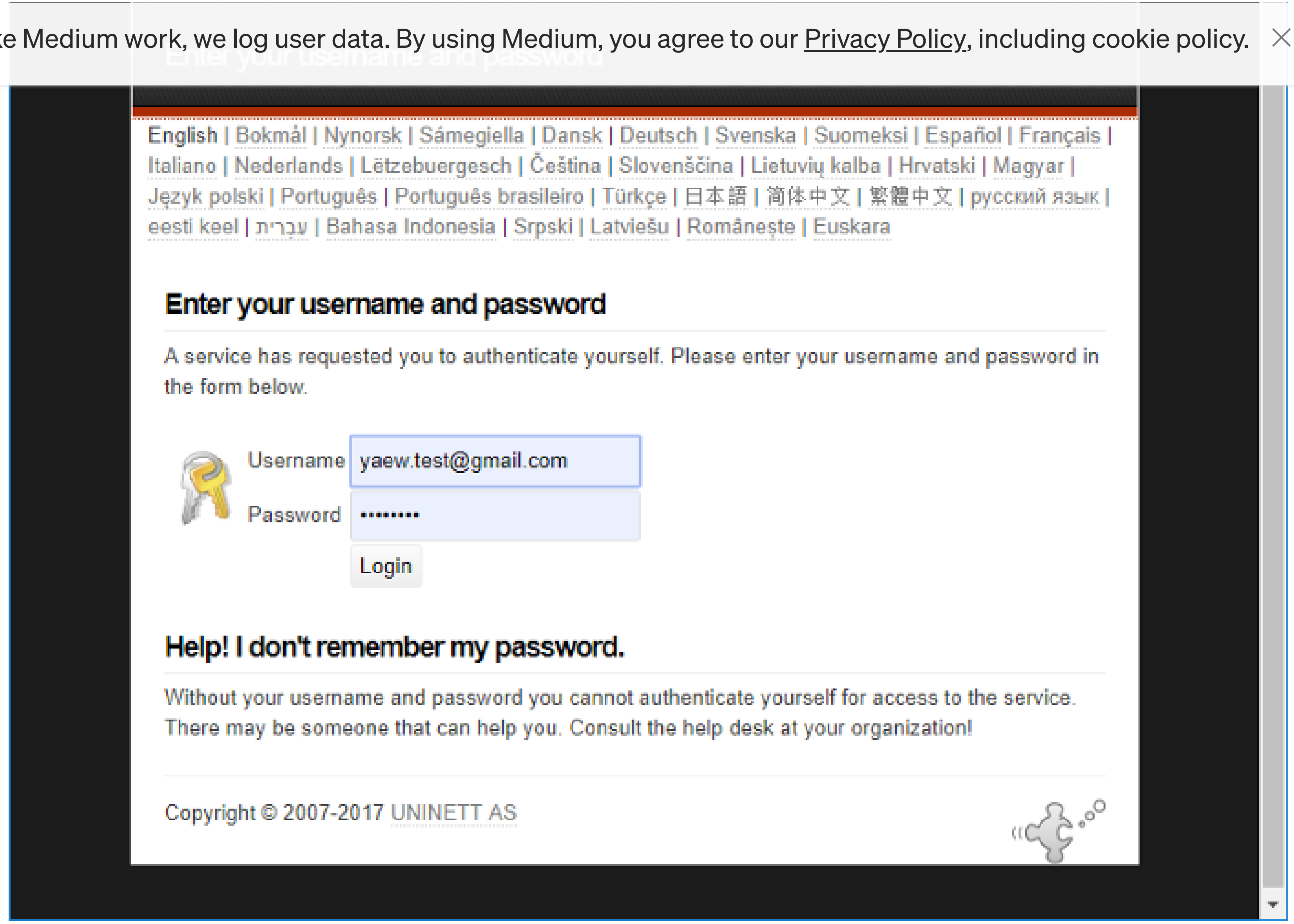
- admin
- example-userpass

Copyright © 2007-2017 UNINETT AS

Then click `example-userpass` link. Opening this URL to your localhost should bring you directly there:



Enter your username and passwo
localhost:8080/simplesaml/module.php/core/loginuserpass...
Paused



English | Bokmål | Nynorsk | Sámegiella | Dansk | Deutsch | Svenska | Suomeksi | Español | Français | Italiano | Nederlands | Lëtzebuergesch | Čeština | Slovenščina | Lietuvių kalba | Hrvatski | Magyar | Język polski | Português | Português brasileiro | Türkçe | 日本語 | 简体中文 | 繁體中文 | русский язык | eesti keel | עִבְרִית | Bahasa Indonesia | Srpski | Latviešu | Românește | Euskara

Enter your username and password

A service has requested you to authenticate yourself. Please enter your username and password in the form below.

Username yaew.test@gmail.com

Password •••••••••

Login

Help! I don't remember my password.

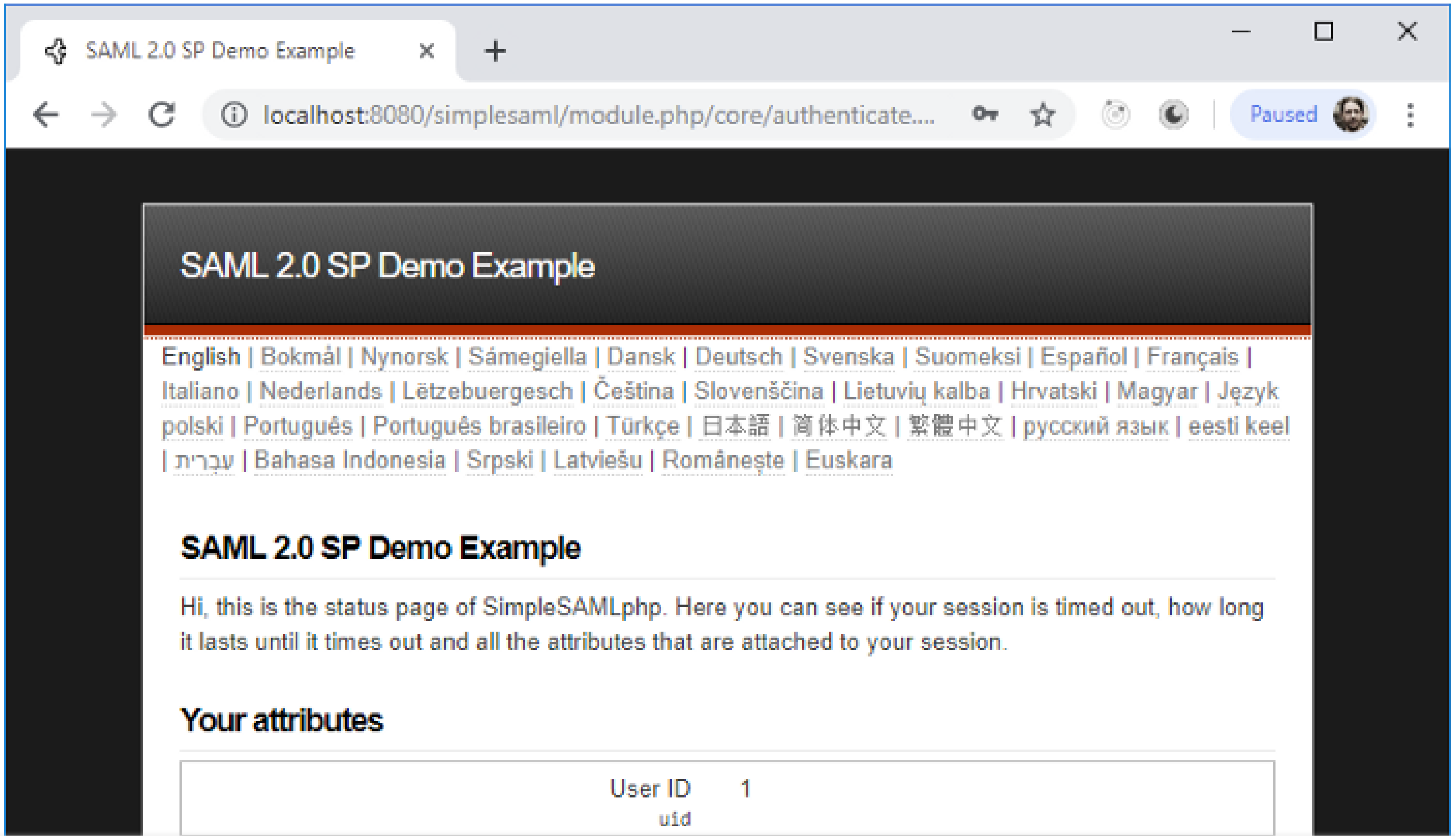
Without your username and password you cannot authenticate yourself for access to the service. There may be someone that can help you. Consult the help desk at your organization!

Copyright © 2007-2017 UNINETT AS

There are two default users created in this app by default:

```
------------------------------------------------------------------
1    | user1     | user1pass | group1 | user1@example.com
2    | user2     | user2pass | group2 | user2@example.com
```

Enter one of these users and click the login button, you should see the user information output to the screen:



SAML 2.0 SP Demo Example

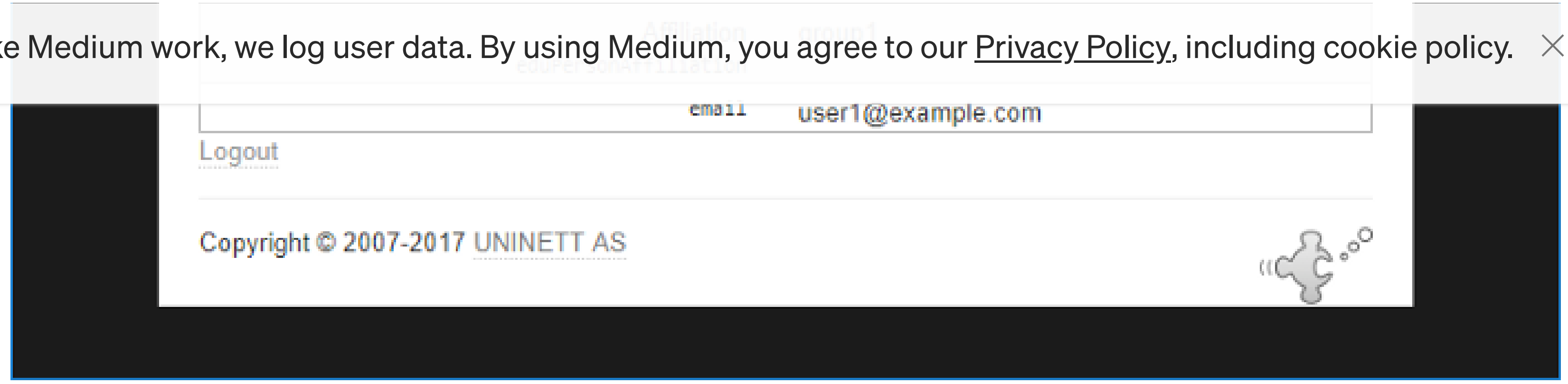
English | Bokmål | Nynorsk | Sámegiella | Dansk | Deutsch | Svenska | Suomeksi | Español | Français | Italiano | Nederlands | Lëtzebuergesch | Čeština | Slovenščina | Lietuvių kalba | Hrvatski | Magyar | Język polski | Português | Português brasileiro | Türkçe | 日本語 | 简体中文 | 繁體中文 | русский язык | eesti keel | עִבְרִית | Bahasa Indonesia | Srpski | Latviešu | Românește | Euskara

SAML 2.0 SP Demo Example

Hi, this is the status page of SimpleSAMLphp. Here you can see if your session is timed out, how long it lasts until it times out and all the attributes that are attached to your session.

Your attributes

User ID uid	1


email user1@example.com

Logout

Copyright © 2007-2017 UNINETT AS

Click around and go back to the login screen. You will not be presented with another login screen until you log out — or until your cookie expires. The IdP is keeping track of your session login and this is independent of the SPs session tracking.

If you think about your day and something that uses single sign on, you'll realize that you don't sign on all that much. Google is a great example. I probably access a dozen or so apps that integrate with a single sign on provider — Google Calendar, Gmail, and YouTube are some examples, but will often go for days without signing back in. This caching mechanism on the IdP side allows me to log in once a day, while still have access to all other services.

Create a Service Provider

With the IdP all ready to go, it is time to create the service provider to integrate with it.

Setup The Node Libraries

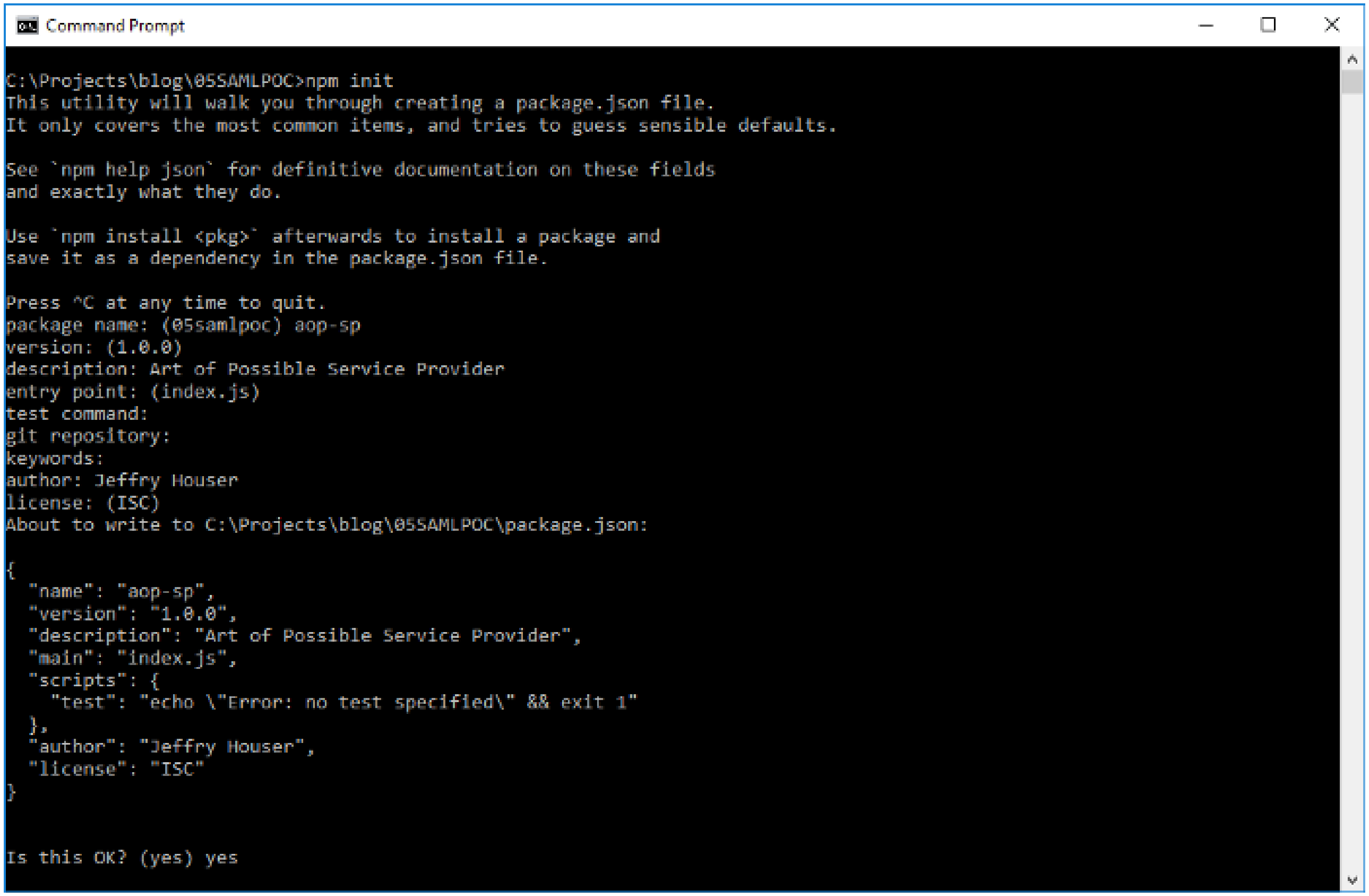
You should already have NodeJS installed if you followed the perquisite sets earlier in the article. Run:

```
npm init
```

In a blank directory to create the project. Enter these values:

- Package Name: aop-sp
- Version: 1.0.0
- Description: Art of Possible Service Provider
- Entry Point: index.js
- Test command: [leave blank]
- Git repository: [leave blank]
- Keywords: leave blank
- Author: Your Name
- License: (ISC)

You should see something like this:

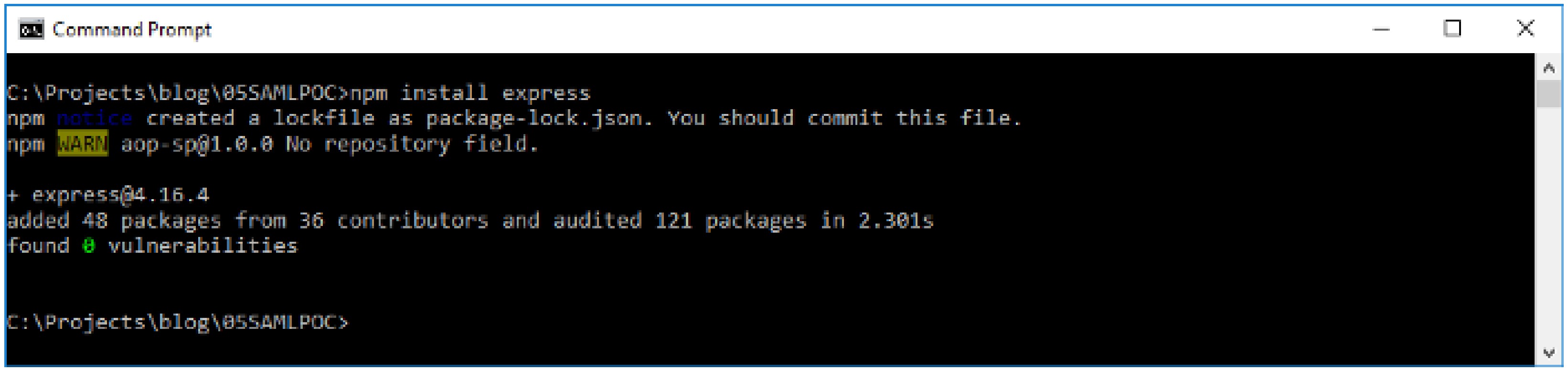


```
C:\Projects\blog\05SAMLPOC>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (05samlpoc) aop-sp
version: (1.0.0)
description: Art of Possible Service Provider
entry point: (index.js)
test command:
git repository:
keywords:
author: Jeffry Houser
license: (ISC)
About to write to C:\Projects\blog\05SAMLPOC\package.json:

{
  "name": "aop-sp",
  "version": "1.0.0",
  "description": "Art of Possible Service Provider",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Jeffry Houser",
  "license": "ISC"
}


Is this OK? (yes) yes
```

Now let's start installing some Node packages. First, install Express:

```
npm install express
```

You should see something like this:

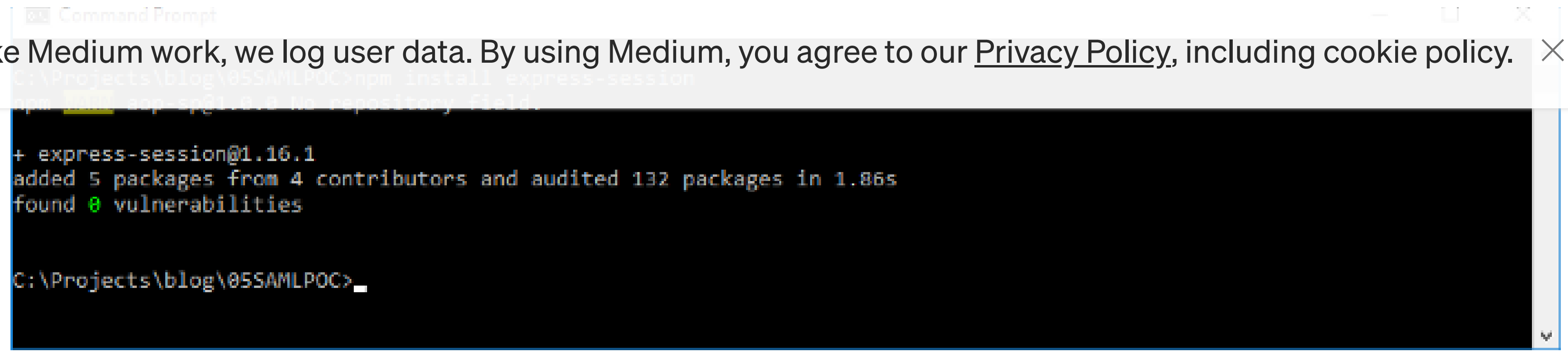

```
C:\Projects\blog\05SAMLPOC>npm install express
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN aop-sp@1.0.0 No repository field.

+ express@4.16.4
added 48 packages from 36 contributors and audited 121 packages in 2.301s
found 0 vulnerabilities


C:\Projects\blog\05SAMLPOC>
```

Express is a web server for NodeJS, and we'll use that as part of our system.

Now install express-session, which allows us to create a server side session associated with a cookie:

```
npm install express-session
```

You should see something like this:



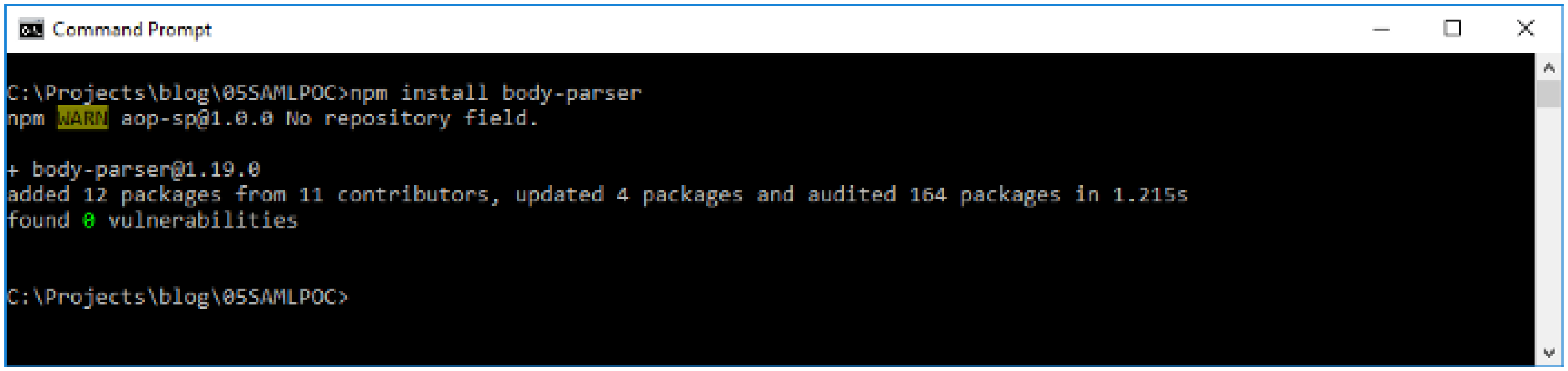
To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

```
C:\Projects\blog\05SAMLPOC>npm install express-session
npm WARN aop-sp@1.0.0 No repository field.

+ express-session@1.16.1
added 5 packages from 4 contributors and audited 132 packages in 1.86s
found 0 vulnerabilities


C:\Projects\blog\05SAMLPOC>_
```

Next we want to install an express body parser

```
npm install body-parser
```

You'll see this:



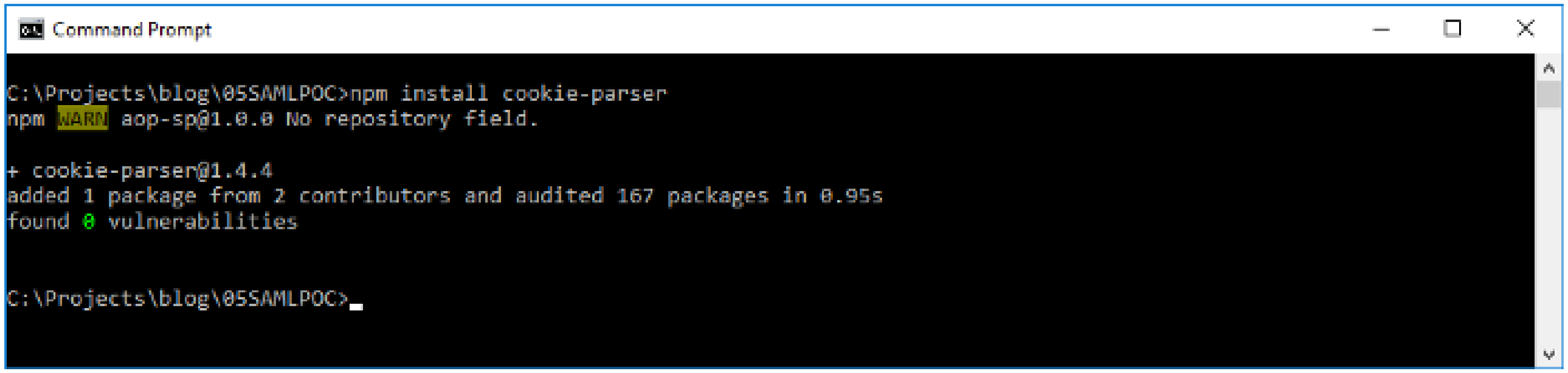
```
C:\Projects\blog\05SAMLPOC>npm install body-parser
npm WARN aop-sp@1.0.0 No repository field.

+ body-parser@1.19.0
added 12 packages from 11 contributors, updated 4 packages and audited 164 packages in 1.215s
found 0 vulnerabilities


C:\Projects\blog\05SAMLPOC>
```

an object instead of trying to deal with raw data as part of the request.

Next, load up the cookie-parser, which copies the cookie header of incoming requests to an object of cookie names:

```
npm install cookie-parser
```

You'll see this:

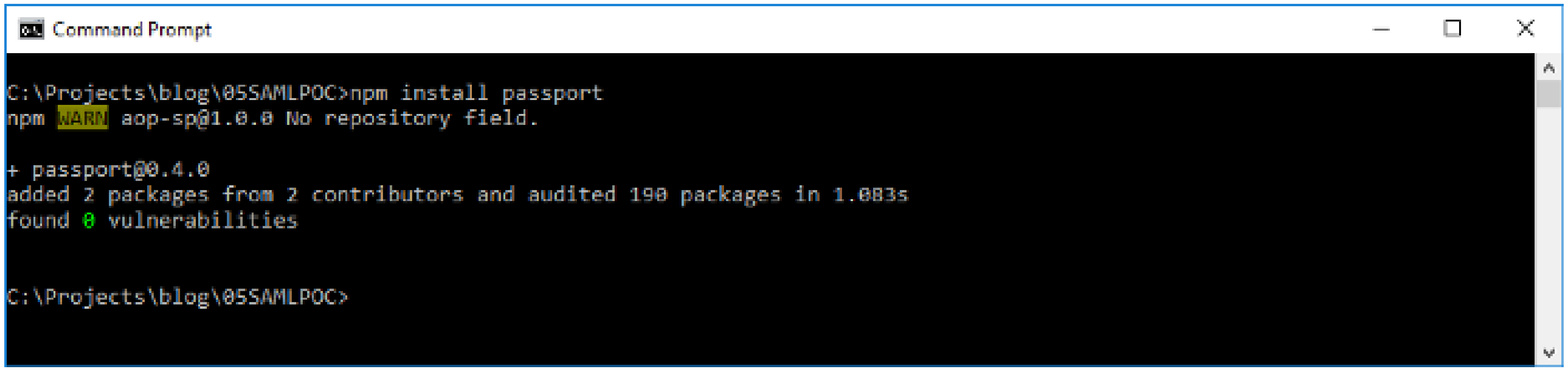


```
C:\Projects\blog\05SAMLPOC>npm install cookie-parser
npm WARN aop-sp@1.0.0 No repository field.

+ cookie-parser@1.4.4
added 1 package from 2 contributors and audited 167 packages in 0.95s
found 0 vulnerabilities


C:\Projects\blog\05SAMLPOC>
```

Next install the Passport Library:

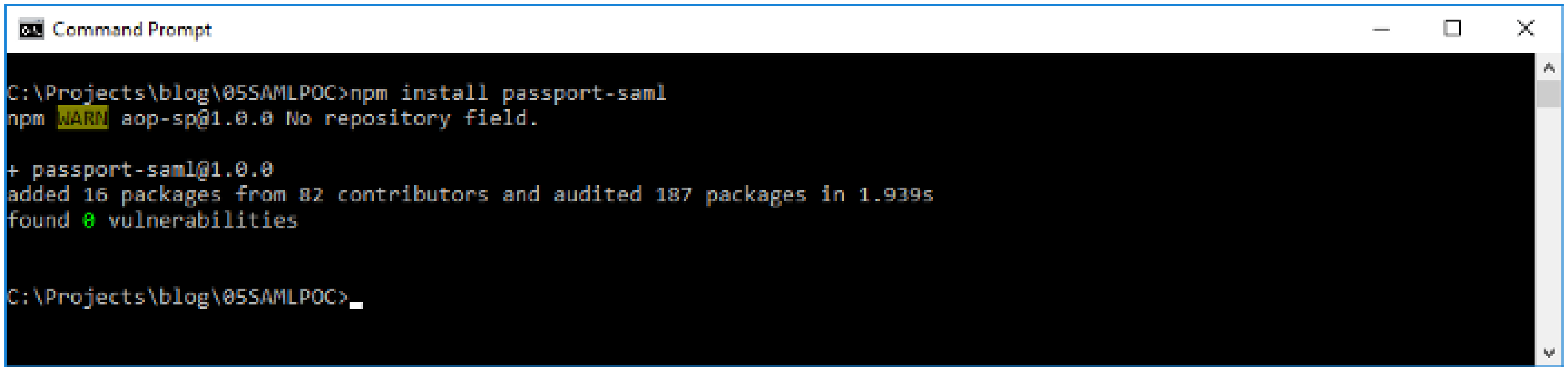
```
npm install passport
```

see this:


Command Prompt

```
C:\Projects\blog\05SAMLPOC>npm install passport
npm WARN aop-sp@1.0.0 No repository field.

+ passport@0.4.0
added 2 packages from 2 contributors and audited 190 packages in 1.083s
found 0 vulnerabilities


C:\Projects\blog\05SAMLPOC>
```

Finally, install passport-saml. This is a SAML plugin to the Passport library.


Command Prompt

```
C:\Projects\blog\05SAMLPOC>npm install passport-saml
npm WARN aop-sp@1.0.0 No repository field.

+ passport-saml@1.0.0
added 16 packages from 82 contributors and audited 187 packages in 1.939s
found 0 vulnerabilities


C:\Projects\blog\05SAMLPOC>
```

While the Passport library provides a framework for handling authentication, it is extensible to allow for different approaches to be plugged into it. These approaches are called strategies, and the passport-saml library is a SAML strategy for Passport. We use it authentication library or process the packets that get returned. This library makes our lives easier by doing that for us.

If you've been following along, you should have a **package.json** that looks something like this:

```
{
  "name": "aop-sp",
  "version": "1.0.0",
  "description": "Art of Possible Service Provider",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Jeffry Houser",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.19.0",
    "cookie-parser": "^1.4.4",
    "express": "^4.16.4",
    "express-session": "^1.16.1",
    "passport": "^0.4.0",
    "passport-saml": "^1.0.0"
  }
}
```

populating that shortly.

Create Certs

We are going to need three certificate files for our application. First, we'll use openSSL to create a public and private cert for our application. We'll encrypt our SAML requests with our private key, and the IdP will use the public key to decrypt them. The IdP will encrypt the responses with its private key, and we'll use its public key to decrypt the responses.

First, let's use OpenSSL to create our own keys. Run this at your console:

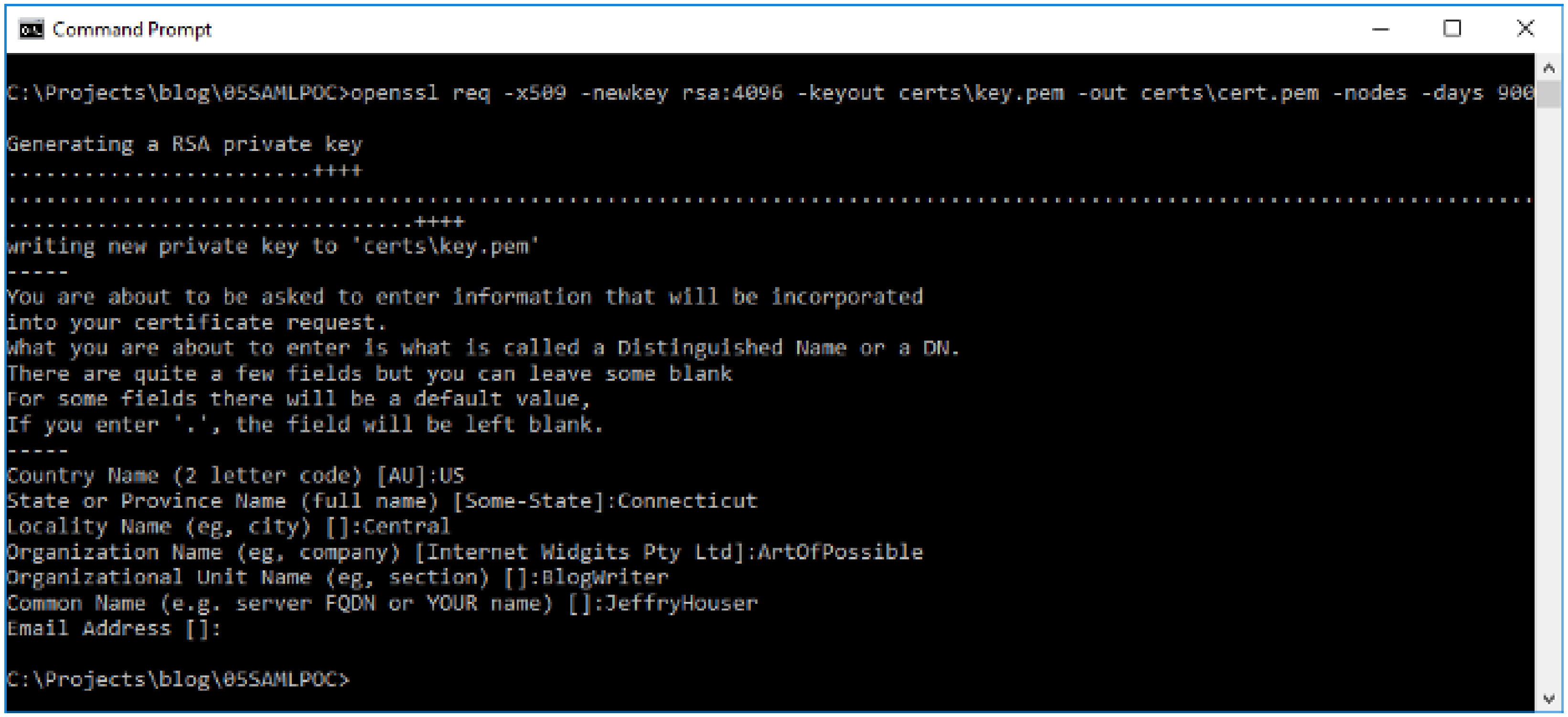
```
openssl req -x509 -newkey rsa:4096 -keyout certs\key.pem
-out certs\cert.pem -nodes -days 900
```

You'll need to create your certs directory before running that command. This command will step you through a wizard asking you for pertinent information to the key generation. Here are the values I entered:

- **Country Name**: US
- **State or Province**: Connecticut

- **Organization Name**: ArtOfPossible
- **Organizational Unit**: BlogWriter
- **Common Name**: JeffryHouser
- **EmailAddress**: [This space intentionally Left Blank]

You should see something like this with the final results:



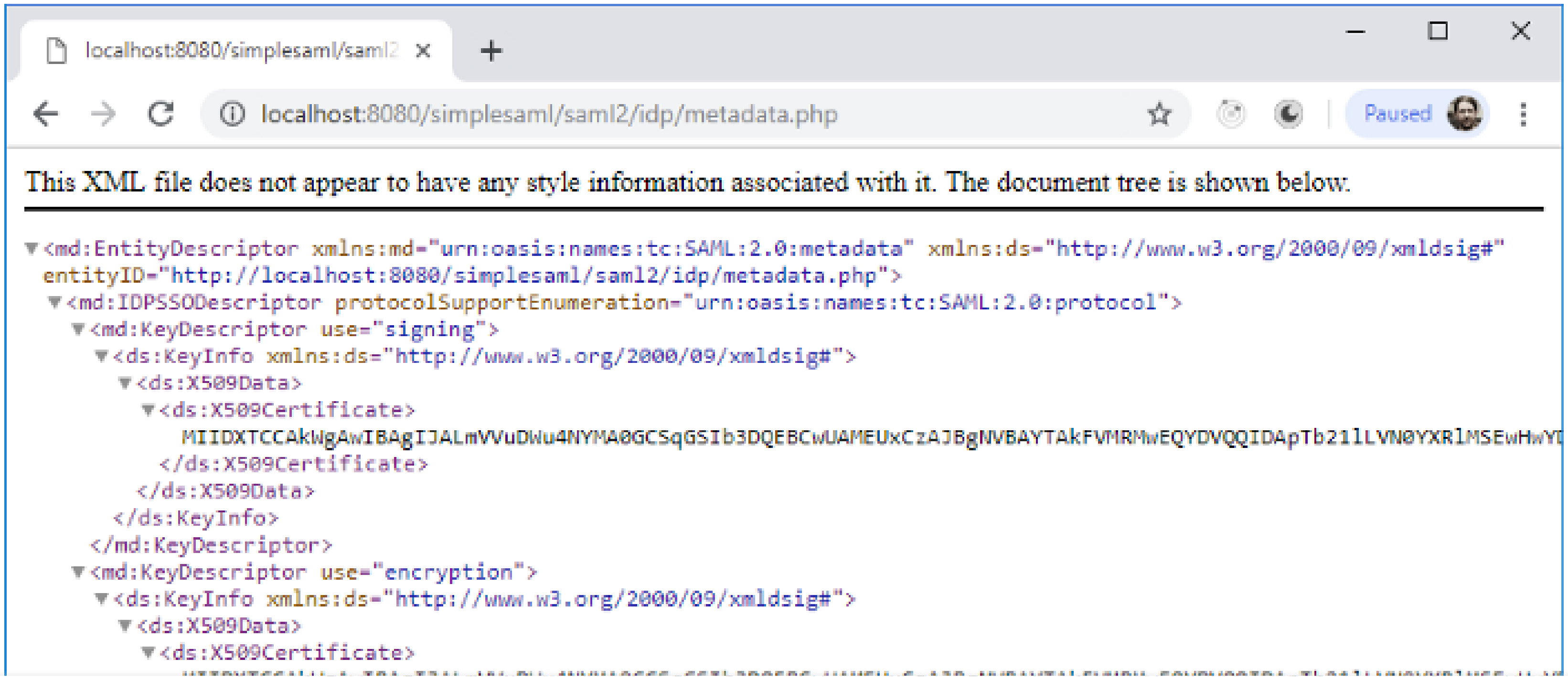
```
C:\Projects\blog\05SAMLPOC>openssl req -x509 -newkey rsa:4096 -keyout certs\key.pem -out certs\cert.pem -nodes -days 900

Generating a RSA private key
........................++++
..........................................................................................................................
..................................++++
writing new private key to 'certs\key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Connecticut
Locality Name (eg, city) []:Central
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ArtOfPossible
Organizational Unit Name (eg, section) []:BlogWriter
Common Name (e.g. server FQDN or YOUR name) []:JeffryHouser
Email Address []:

C:\Projects\blog\05SAMLPOC>
```

- **cert.pem**: Your Public cert
- **key.pem**: Your Private cert.

We'll reference these from our code when creating our service provider. We need one more cert, the IdP's public cert. To get that, open up the IdP's metadata page:

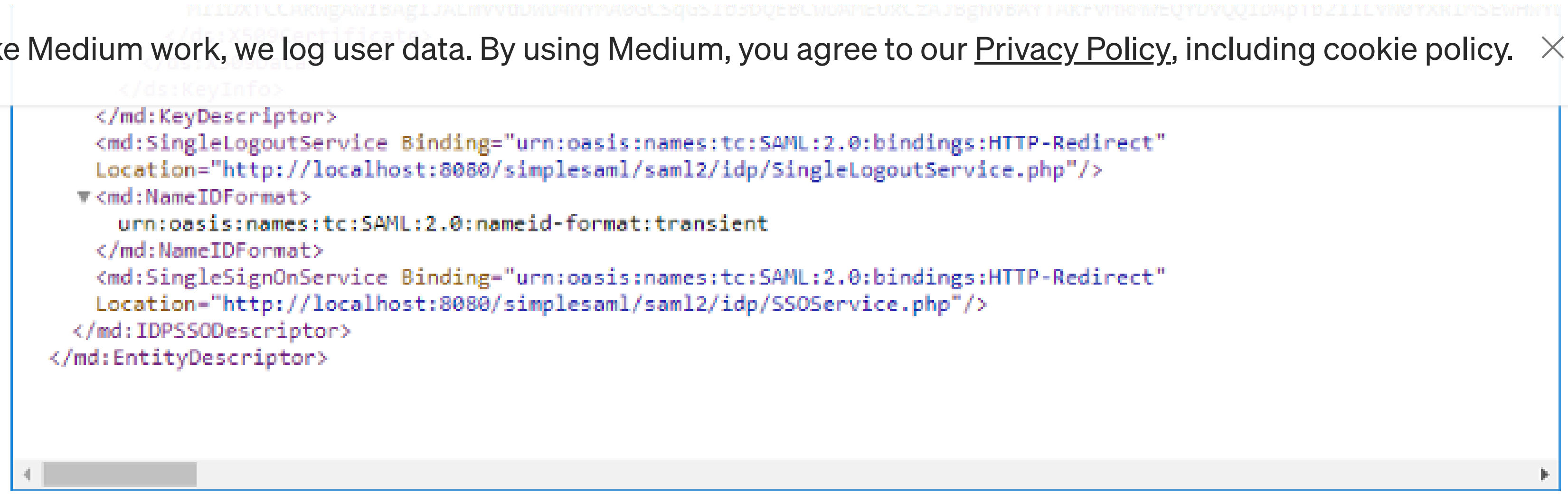
http://localhost:8080/simplesaml/saml2/idp/metadata.php

You should see:


localhost:8080/simplesaml/saml2
localhost:8080/simplesaml/saml2/idp/metadata.php
Paused

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 entityID="http://localhost:8080/simplesaml/saml2/idp/metadata.php">
  ▼<md:IDPSSODescriptor protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    ▼<md:KeyDescriptor use="signing">
      ▼<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        ▼<ds:X509Data>
          ▼<ds:X509Certificate>
             MIIDXTCCAkWgAwIBAgIJALmVVuDWu4NYMA0GCSqGSIb3DQEBCwUAMEUxCzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwY
           </ds:X509Certificate>
         </ds:X509Data>
       </ds:KeyInfo>
     </md:KeyDescriptor>
    ▼<md:KeyDescriptor use="encryption">
      ▼<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        ▼<ds:X509Data>
          ▼<ds:X509Certificate>
```


```
    </md:KeyDescriptor>
    <md:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
    Location="http://localhost:8080/simplesaml/saml2/idp/SingleLogoutService.php"/>
  ▼<md:NameIDFormat>
      urn:oasis:names:tc:SAML:2.0:nameid-format:transient
    </md:NameIDFormat>
    <md:SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
    Location="http://localhost:8080/simplesaml/saml2/idp/SSOService.php"/>
  </md:IDPSSODescriptor>
</md:EntityDescriptor>
```

Look for the X509Certificate tag in the XML and copy it to a file named **idp_key.pem** in your certs directory.

Setup Express Web Sever

Let’s set up Express. Open up the empty **index.js**. Start with some imports:

```
var express = require("express");
var session = require('express-session');
var bodyParser = require('body-parser');
var cookieParser = require('cookie-parser');
```

This imports the express web server and three plugins — express-session, body-parser, and cookie-parser.

```
var app = express();
```

Now, we tell the Express instance to use the other plugs. First, the **cookieParser:**

```
app.use(cookieParser());
```

Cookies will be required to tell whether the user is authenticated or not. This will be done behind the scenes by the passport library.

Now set up the **bodyParser:**

```
app.use(bodyParser.urlencoded({ extended: false }))
app.use(bodyParser.json())
```

The body parser can turn the body text of a URL request into a simple object for us to access. The **urlencoded()** command will handle `application/x-www/form-urlencoded` values. The **json()** command will take care of any JSON values.

```
app.use(session({secret: 'secret',
                 resave: false,
                 saveUninitialized: true,}));
```

The secret value is used to sign a sessionID cookie. The sessionID will reference the server-side session. We can use any value we want for the secret key, but for the purposes of this sample I made it simple. The **resave** value determines whether to save the session value back into the session store after every request, even if it was not changed. Typically there is no need to do this. The **saveUninitailized** value is set to true. This means that a session is always saved after it was created even if it did not change.

Now let’s create a handler for the root of our application:

```
app.get('/',
    function(req, res) {
        res.send('Test Home Page');
    }
);
```

This is a get() request on the express **app** variable. It looks for the root directory, '/'. The way that express works is that each request is a collection of functions. The function accepts the request and response arguments, abbreviated to **req** and **res** respectively. In this case, we only run a single function which returns the text 'Test Home Page'. Later we'll run functions to validate users.

Finally, add some code to start the server:

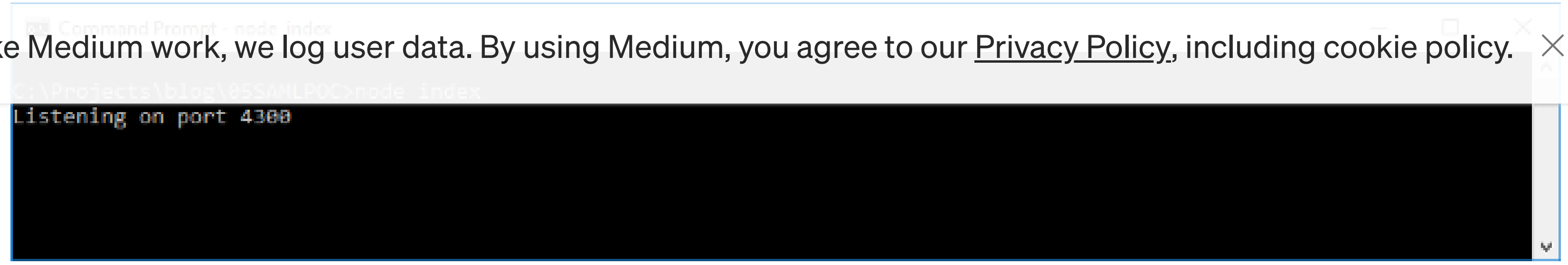
```
var server = app.listen(4300, function () {
    console.log('Listening on port %d', server.address().port)
});
```

This will listen to port 4300, and show something on the console to prove that is listening.

Run the server:

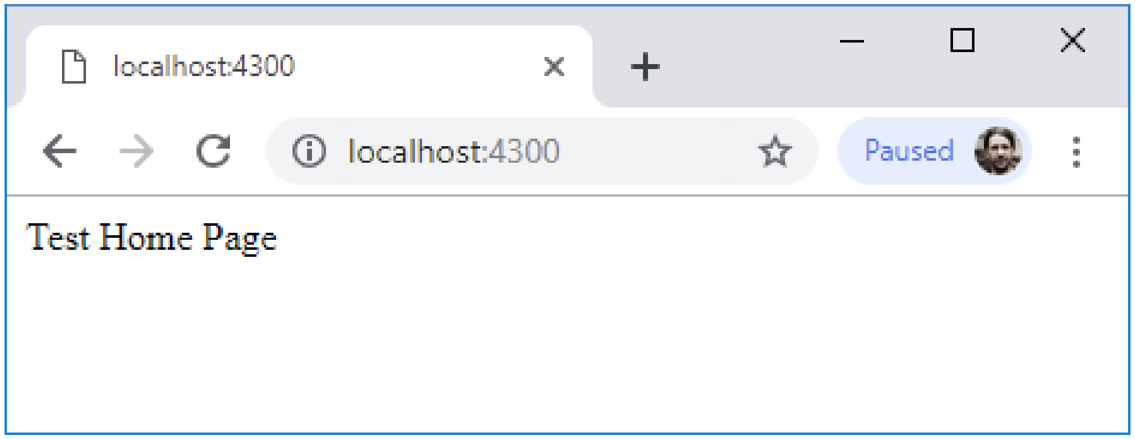
```
node index
```

You should see:



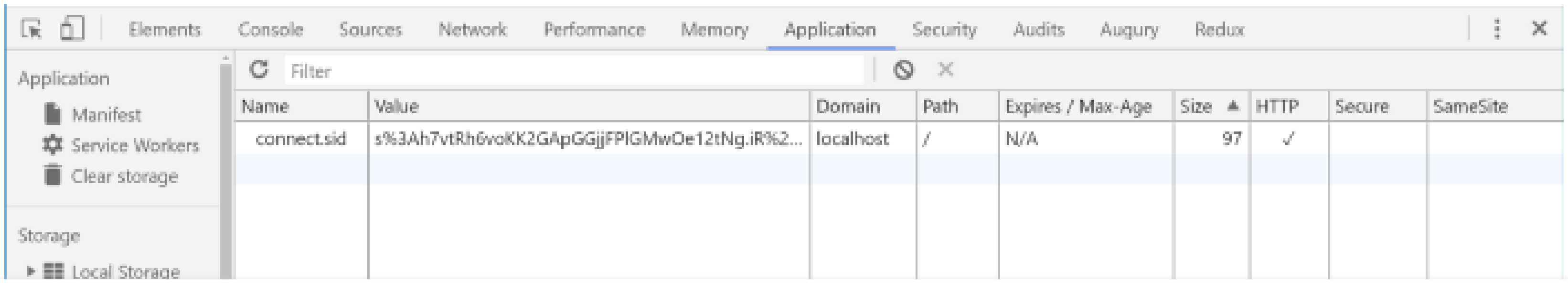
```
Listening on port 4300
```

Load the root in your browser at http://localhost:4300 :

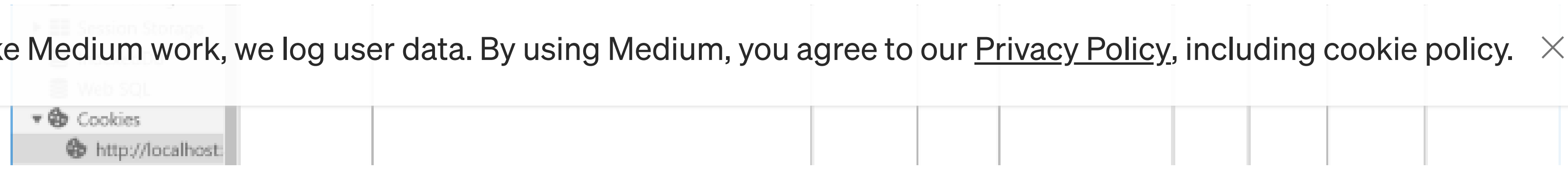


Test Home Page

Pop open the web developer tools to look at your cookies. Go to the Application tab and expand cookies under storage:



Name	Value	Domain	Path	Expires / Max-Age	Size	HTTP	Secure	SameSite
connect.sid	s%3Ah7vtRh6voKK2GApGGjjFPlGMwOe12tNg.iR%2...	localhost	/	N/A	97	✓		



You should see the **connect.sid** cookie which was created by the express server. This is the session identifier that the server uses to track you between browser requests. Your browser is successfully hooked up to a server session, even though no data is stored in it yet.

Configure Simple-SAML

Now we're ready to setup the Passport and SAML libraries. First load the libraries:

```
var passport = require('passport');
var saml = require('passport-saml');
var fs = require('fs');
```

I've already mentioned passport and passport-saml, but I also add the fs library, which will let us access the file system. We'll use that to load our certs from disk into the SAML configuration.

Passport requires that we add functions to serialize and deserialize the user, so that is the first thing we'll do:

```
passport.serializeUser(function(user, done) {
    console.log('------------------------------');
    console.log('serialize user');
    console.log(user);
    console.log('------------------------------');
    done(null, user);
});

passport.deserializeUser(function(user, done) {
    console.log('------------------------------');
    console.log('deserialize user');
    console.log(user);
    console.log('------------------------------');
    done(null, user);
});
```

These functions are default functions that just output the user to the console, which is a great debugging tool.

We need to set up a **samlStrategy**, so that Passport knows how to create requests and process the login. Start with this:

```
var samlStrategy = new saml.Strategy({
  // config options here
}, function(profile, done) {
    return done(null, profile);
});
```

The **saml.Strategy()** accepts two arguments. The first is a configuration object, which I left blank for the moment, and the second is a function which processes the user. The first argument into the function is a **profile** object, and the second is **done**, a callback. For our purposes, we are just loading executing the callback and sending it the profile object unchanged. If we needed to do more functionality, such as load application specific permissions from a database, this could be done here.

Now, let's populate the configuration object with values. I decided to drop these in one by one so I could explain each one:

```
callbackUrl: 'http://localhost/login/callback',
```

The **callbackUrl** is a URL in our application — the service provider — where the IdP will post back to after a successful user authentication. We haven't created this URL yet, but we will.

```
entryPoint:
'http://localhost:8080/simplesaml/saml2/idp/SSOService.php',
```

The entryPoint is the URL in the IdP that we will send our request to in order to let the user authenticate. When we ran the docker image to create our local IdP, we passed this value into it as a configuration option named **SIMPLESAMLPHP_SP_ASSERTION_CONSUMER_SERVICE.**

Now the issuer:

```
issuer: 'saml-poc',
```

The **issuer** is a globally unique identifier for our application. When we ran the docker image, we passed this value into it as a configuration option named **SIMPLESAMLPHP_SP_ENTITY_ID**.

```
identifierFormat: null,
```

The **identifierFormat** is a specific format you can request from the IdP. We're leaving it null here, but most likely your IdP administrators will provide a value that you must enter.

Now set up the keys:

```
decryptionPvk: fs.readFileSync(__dirname + '/certs/key.pem', 'utf8'),
privateCert: fs.readFileSync(__dirname + '/certs/key.pem', 'utf8'),
```

The **decryptionPvK** and **privateCert** both refer to the local private key we generated. They are used to encrypt the authentication request before we send it to the IdP. This is where I'm using the fs library to load the cert from disc.

```
validateInResponseTo: false,
```

The **validateInResponseTo** value will determine if the incoming SAML responses need to be validated or not. I set it to false for simplicity in our sample.

```
disableRequestedAuthnContext: true
```

The **disableRequestdAuthnContext** is another Boolean value. This can be helpful when authenticating against an Active Directory Server.

That completes our **samlStrategy** configuration object.

```
passport.use('samlStrategy', samlStrategy);
```

Simple enough, now initialize passport:

```
app.use(passport.initialize({}));
app.use(passport.session({}));
```

The **session()** is middleware that allows for persistent login — AKA keeping track of users.

Create Login Routes

Let’s create the login handler. It is pretty simple:

```
app.get('/login',
    function (req, res, next) {
        console.log('-----------------------------');
        console.log('/Start login handler');
        next();
    },
    passport.authenticate('samlStrategy'),
);
```

I'm using a **get()** handler on the app variable, and the value is '/login'. That means when I load `http://localhost:4300/login` it will run the functions, one after each other. The first function just outputs to the console that the log is executed, and then calls **next()**. The **next()** function is a reference to the next handler function. For the next handler function, we are just telling the passport library to authenticate using the 'samlStrategy'. This will redirect to the IdP which will handle login, and post results back to a `login/callback` handler.

Here is the callback handler:

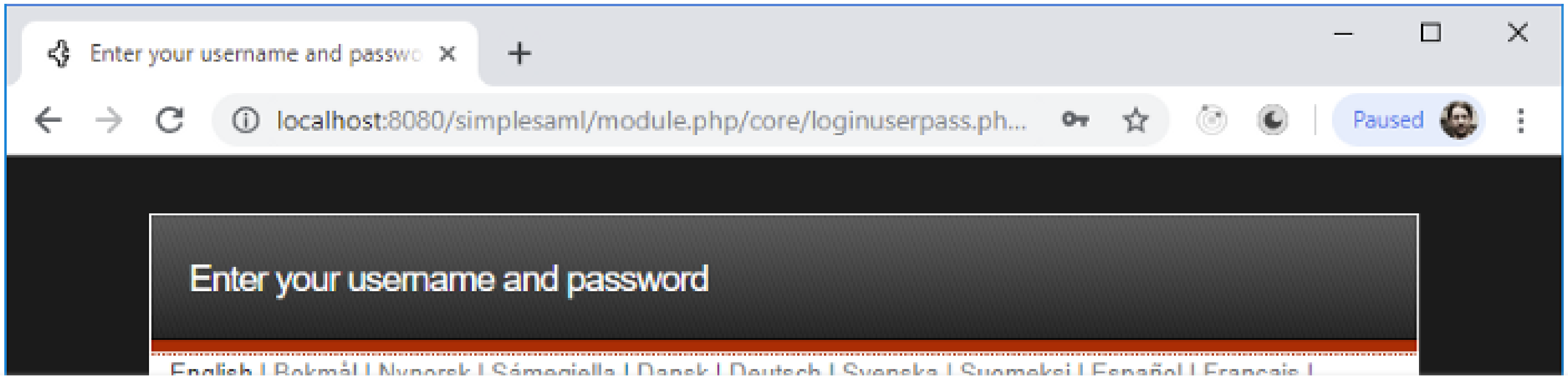
```
app.post('/login/callback',
    function (req, res, next) {
        console.log('-----------------------------');
        console.log('/Start login callback ');
        next();
    },
    passport.authenticate('samlStrategy'),
    function (req, res) {
        console.log('-----------------------------');
        console.log('login call back dumps');
        console.log(req.user);
        console.log('-----------------------------');
        res.send('Log in Callback Success');
```

This calls the **post()** method on the express instance, **app**. The URL is the first argument of the method, '/login/callback/'. First there is a function, which just logs the currently running request; and calls **next()** so that the next function can run. The next function is the **passport.authentication()** call. This is the same code that we had in login, but here in the callback it sees that we have a return value from the IdP and processes it by calling the **serializeUser()** function we set up earlier. Then it calls the next function, which outputs the user returned from the service.

Try this. First load up:

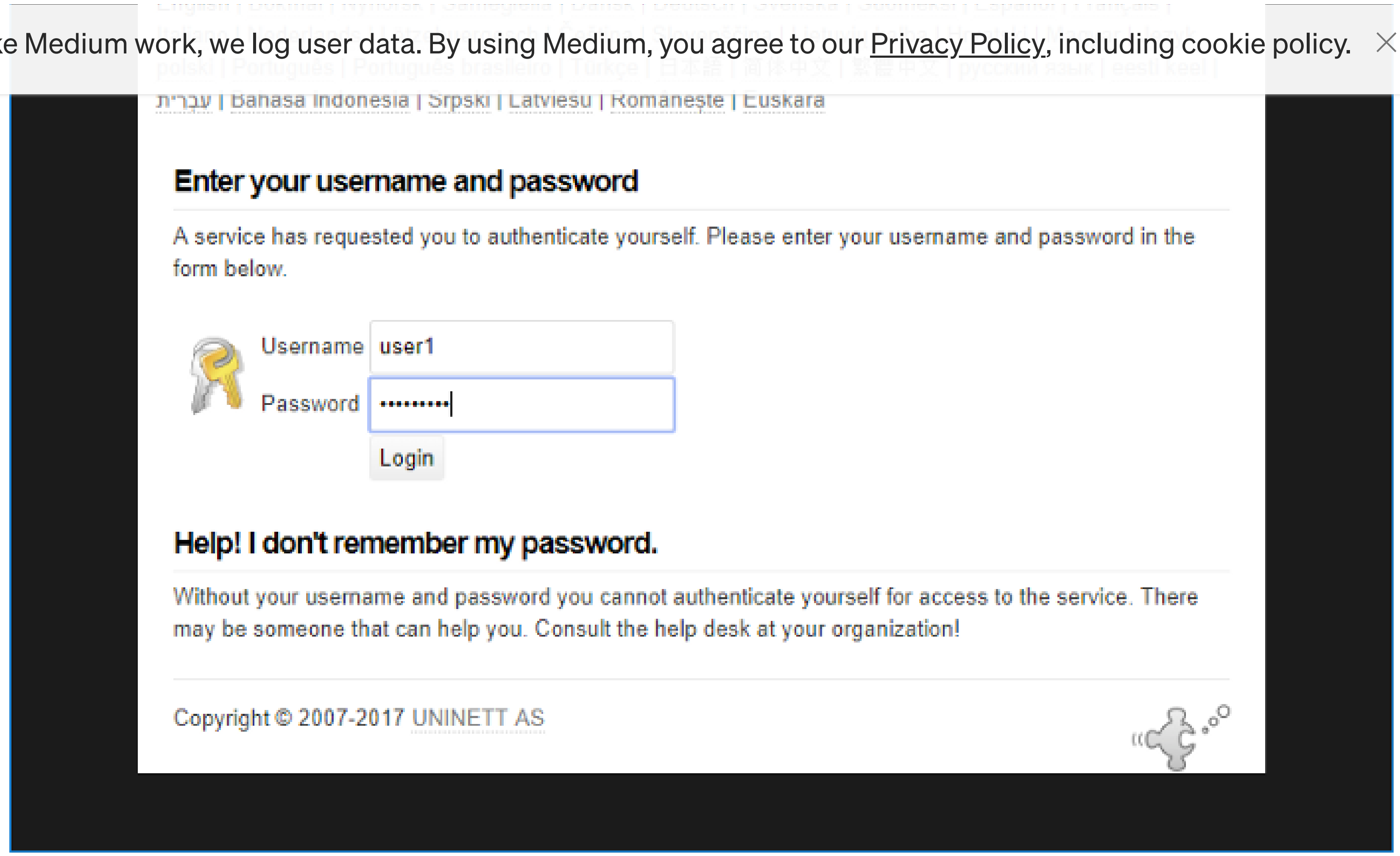
http://localhost:4300/login

You won't see anything, but you'll automatically be redirected to the IdP login screen:


Enter your username and passwo
localhost:8080/simplesaml/module.php/core/loginuserpass.ph...
Paused

Enter your username and password

English | Bokmål | Nynorsk | Sámegiella | Dansk | Deutsch | Svenska | Suomeksi | Español | Français |



עברית | Bahasa Indonesia | Srpski | Latviesu | Romaneste | Euskara

Enter your username and password

A service has requested you to authenticate yourself. Please enter your username and password in the form below.

Username user1

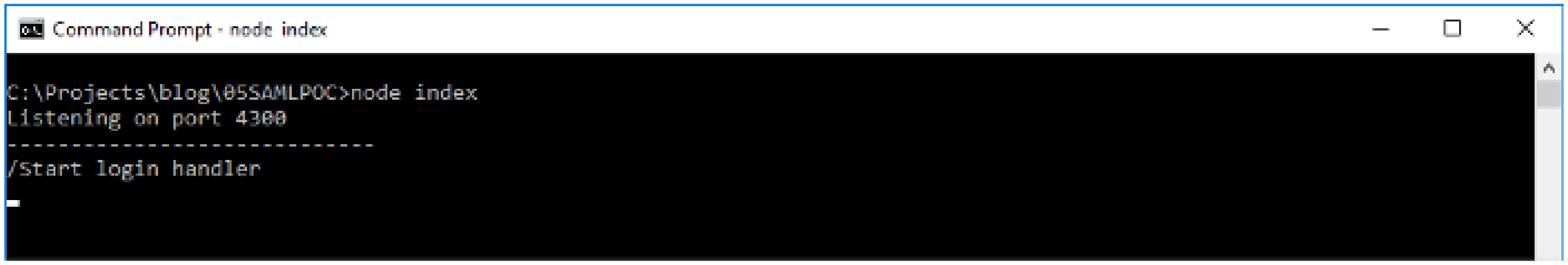
Password ••••••••••

Help! I don't remember my password.

Without your username and password you cannot authenticate yourself for access to the service. There may be someone that can help you. Consult the help desk at your organization!

Copyright © 2007-2017 UNINETT AS

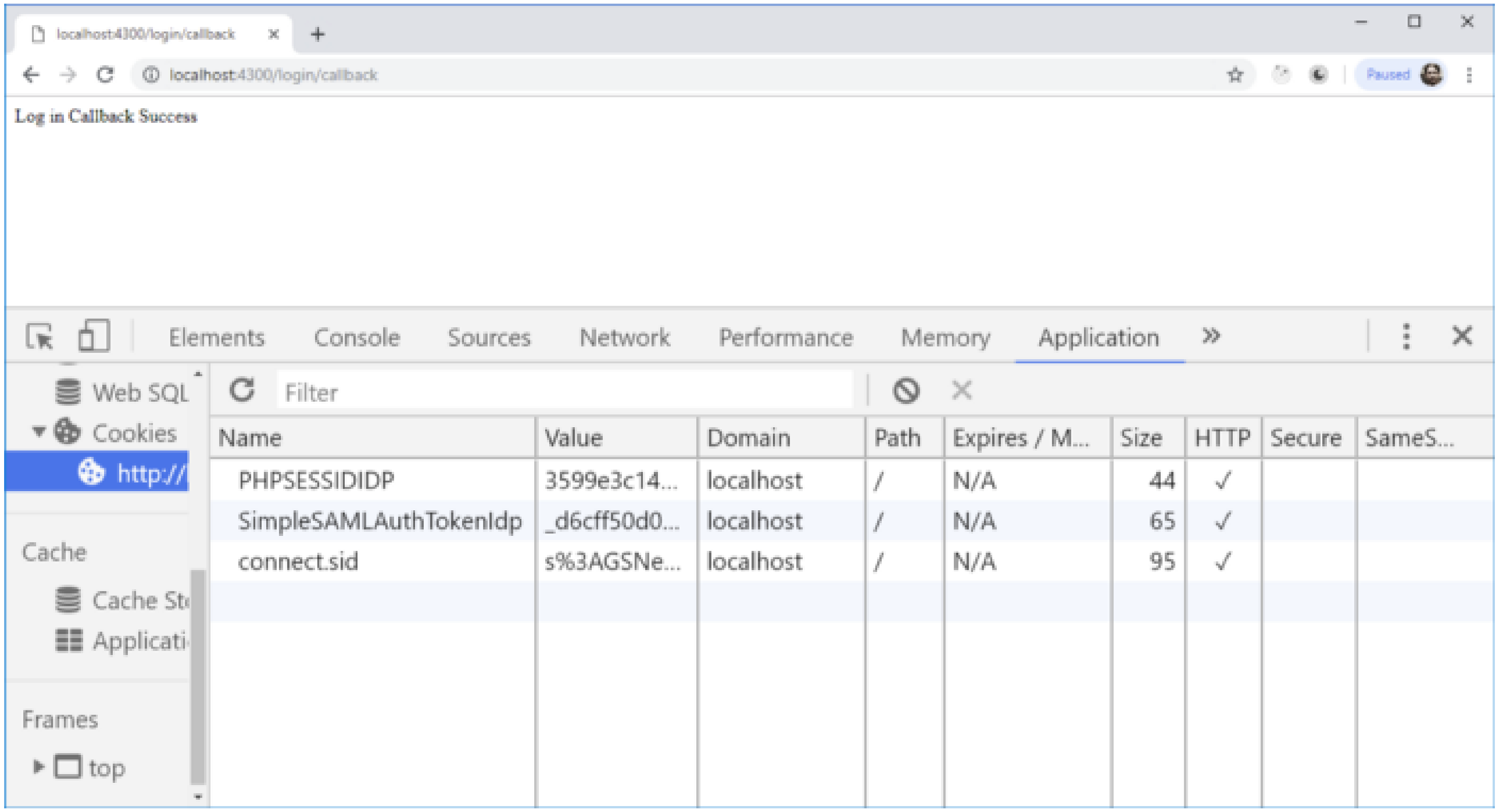
Look at your web server console:



```
C:\Projects\blog\05SAMLPOC>node index
Listening on port 4300
-----------------------------
/Start login handler
```

You'll see that the login handler was properly hit before the redirect.

Now Enter user1 and user1pass and click Login:



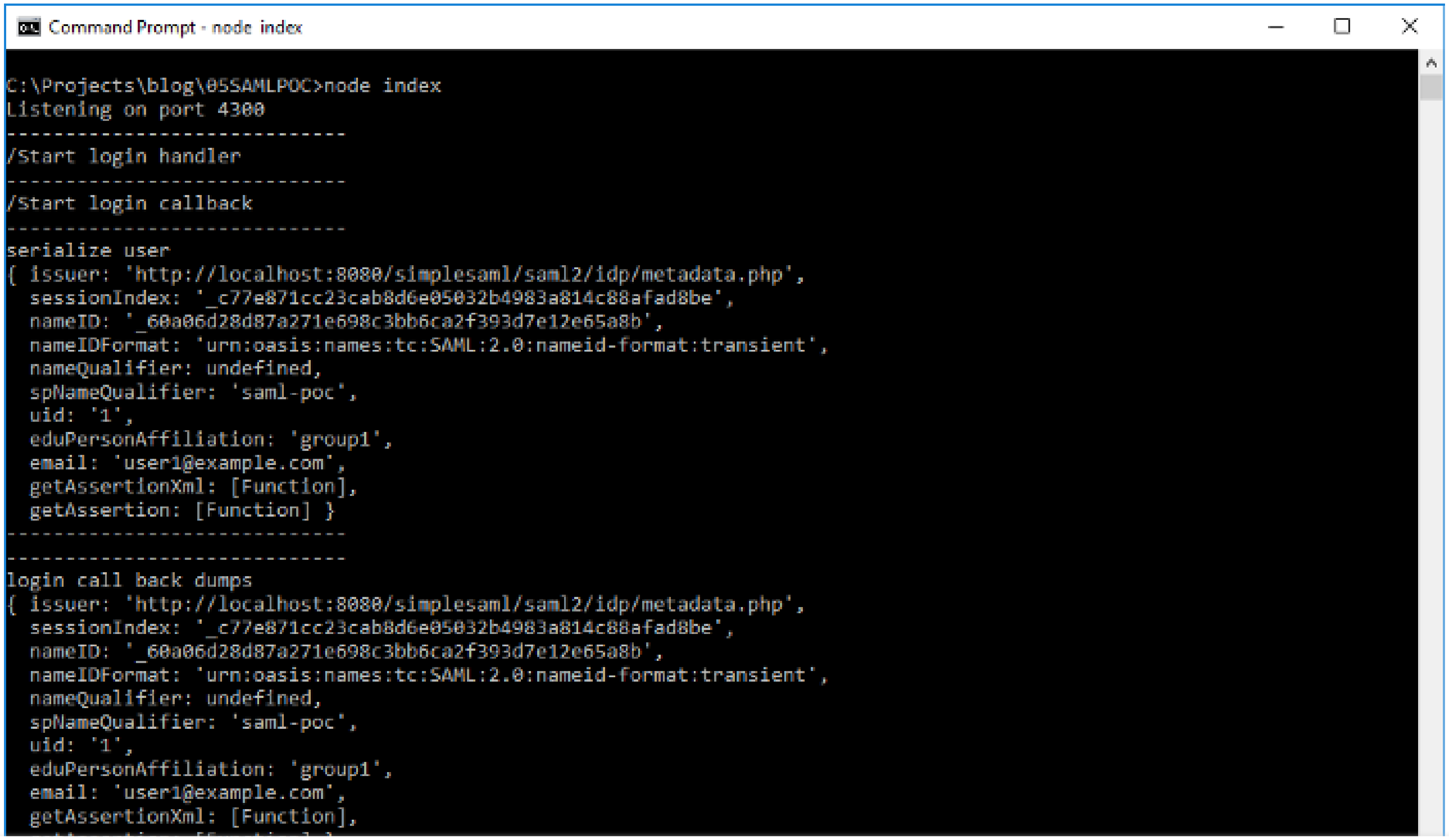
Log in Callback Success

Name	Value	Domain	Path	Expires / M...	Size	HTTP	Secure	SameS...
PHPSESSIDIDP	3599e3c14...	localhost	/	N/A	44	✓		
SimpleSAMLAuthTokenIdp	_d6cff50d0...	localhost	/	N/A	65	✓		
connect.sid	s%3AGSNe...	localhost	/	N/A	95	✓		

You can see that the "Login in CallBack Success" is loaded in the page's body. Checking the cookies in the web developer tools you see three:

- **SimpleSAMLAuthTokenIdp**: This is a User identifier set by the IdP.
- **connect.sid**: This is the session identification token set by our express-session plugin.

Check out the console:


Command Prompt - node index

```
C:\Projects\blog\05SAMLPOC>node index
Listening on port 4300
-----------------------------
/Start login handler
-----------------------------
/Start login callback
-----------------------------
serialize user
{ issuer: 'http://localhost:8080/simplesaml/saml2/idp/metadata.php',
  sessionIndex: '_c77e871cc23cab8d6e05032b4983a814c88afad8be',
  nameID: '_60a06d28d87a271e698c3bb6ca2f393d7e12e65a8b',
  nameIDFormat: 'urn:oasis:names:tc:SAML:2.0:nameid-format:transient',
  nameQualifier: undefined,
  spNameQualifier: 'saml-poc',
  uid: '1',
  eduPersonAffiliation: 'group1',
  email: 'user1@example.com',
  getAssertionXml: [Function],
  getAssertion: [Function] }
-----------------------------
-----------------------------
login call back dumps
{ issuer: 'http://localhost:8080/simplesaml/saml2/idp/metadata.php',
  sessionIndex: '_c77e871cc23cab8d6e05032b4983a814c88afad8be',
  nameID: '_60a06d28d87a271e698c3bb6ca2f393d7e12e65a8b',
  nameIDFormat: 'urn:oasis:names:tc:SAML:2.0:nameid-format:transient',
  nameQualifier: undefined,
  spNameQualifier: 'saml-poc',
  uid: '1',
  eduPersonAffiliation: 'group1',
  email: 'user1@example.com',
  getAssertionXml: [Function],
```

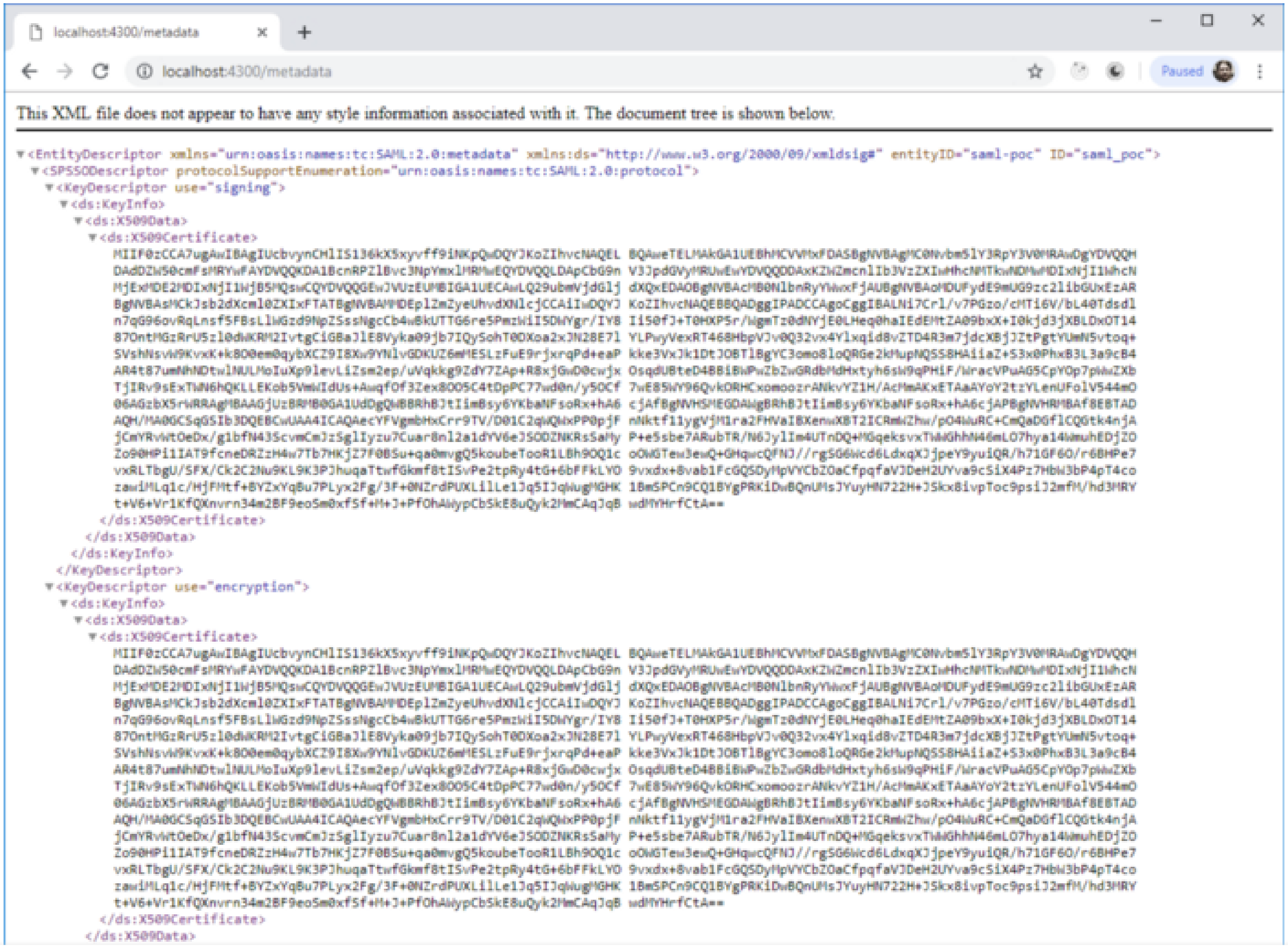
You see after the initial login handler was run; the login callback was run. The **serializeUser()** dumped the user information out to the console; and then again the callback URL dumped out the same user info. This app demonstrates that the login succeeds even if we aren't doing anything with it yet. The information you get back in the user object depends primarily on what the IdP is programmed to send you.

Create our own metadata link

You may remember that the IdP had a metadata link. We used that to get the public key we passed into the cert option of our **samlStrategy** variable. We can create our own metadata route to provide that information to the IdP we are integrating with:

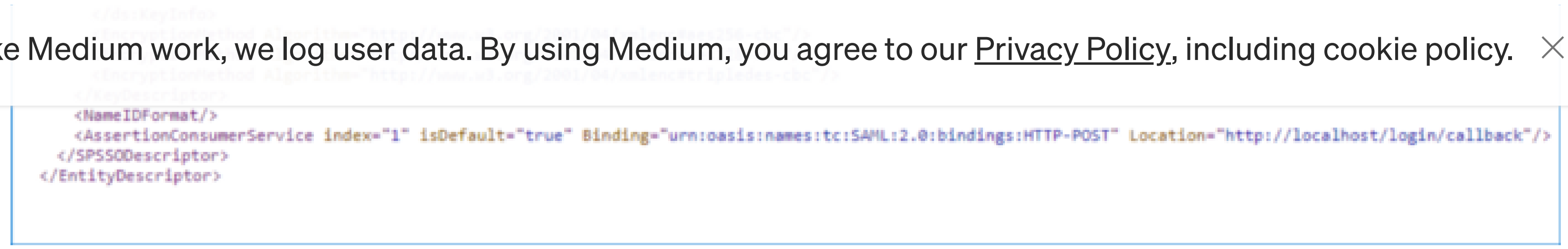
```
app.get('/metadata',
    function(req, res) {
        res.type('application/xml');
        res.status(200).send(
          samlStrategy.generateServiceProviderMetadata(
             fs.readFileSync(__dirname + '/certs/cert.pem', 'utf8'),
             fs.readFileSync(__dirname + '/certs/cert.pem', 'utf8')
          )
        );
    }
);
```

This creates a get request for metadata, and we use the **generateServiceProviderMetadata()** to generate this pages XML. It outputs the public cert in utf8 format. Reload the app with the metadata in here:



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata" xmlns:ds="http://www.w3.org/2000/09/xmldsig#" entityID="saml-poc" ID="saml_poc">
  ▼<SPSSODescriptor protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    ▼<KeyDescriptor use="signing">
      ▼<ds:KeyInfo>
        ▼<ds:X509Data>
          ▼<ds:X509Certificate>
              MIIF0zCCA7ugAwIBAgIUcbvynCHlIS136kX5xyvff9iNKpQwDQYJKoZIhvcNAQEL BQAweTELMAkGA1UEBhMCVVMxFDASBgNVBAgMC0Nvbm5lY3RpY3V0MRAwDgYDVQQH
              DAdDZW50cmFsMRYwFAYDVQQKDA1BcnRPZl0vc3NpYmxlMRMwEQYDVQQLDApCbG9n V3JpdGVyMRUwEwYDVQQDDAxKZWZmcnlIb3VzZXIwHhcNMTkwNDMwMDIxNjI1WhcN
              MjExMDE2MDIxNjI1WjB5MQswCQYDVQQGEwJVUzEUMBIGA1UECAwLQ29ubmVjdGlj dXQxEDAOBgNVBAcMB0NlbnRyYWwxFjAUBgNVBAoMDUFydE9mUG9zc2libGUxEzAR
              BgNVBAsMCkJsb2dXcml0ZXIxFTATBgNVBAMMDEplZmZyeUhvdXNlcjCCAiIwDQYJ KoZIhvcNAQEBBQADggIPADCCAgoCggIBALNi7Crl/v7PGzo/cMTi6V/bL40Tdsdl
              n7qG96ovRqLnsf5FBsLlWGzd9NpZSssNgcCb4wBkUTTG6re5PmzWiISDWYgr/IYB Ii50fJ+T0HXP5r/WgmTz0dNYjE0LHeq0haIEdEMtZA09bxX+I0kjd3jXBLDxOT14
              87OntMGzRrU5zl0dWKRM2IvtgCiGBaJlE8Vyka09jb7IQySohT0DXoa2xJN28E7l YLPwyVexRT468HbpVJv0Q32vx4YlxqId8vZTD4R3m7jdcXBjJZtPgtYUmN5vtoq+
              SVshNsvW9KvxK+k8O0em0qybXCZ9I8Xw9YNlvGDKUZ6mMESLzFuE9rjxrqPd+eaP kke3VxJk1DtJO8TlBgYC3omo8loQRGe2kMupNQSS8HAiiaZ+S3x0PhxB3L3a9cB4
              AR4t87umNhNDtwlNULMoIuXp9levLiZsm2ep/uVqkkg9ZdY7ZAp+R8xjGwD0cwjx OsqdUBteD4BBi8WPwZbZwGRdbMdHxtyh6sW9qPHiF/WracVPuAG5CpYOp7pWwZXb
              TjIRv9sExTWN6hQKLLEKob5VmWIdUs+AwqfOf3Zex8OO5C4tDpPC77wd0n/y5OCf 7wE85WY96QvkORHCxomoozrANkvYZ1H/AcMmAKxETAaAYoY2tzYLenUFolV544mO
              06AGzbX5rWRRAgMBAAGjUzBRMB0GA1UdDgQWBBRhBJtIimBsy6YKbaNFsoRx+hA6 cjAfBgNVHSMEGDAWgBRhBJtIimBsy6YKbaNFsoRx+hA6cjAPBgNVHRMBAf8EBTAD
              AQH/MA0GCSqGSIb3DQEBCwUAA4ICAQAecYFVgmbHxCrr9TV/D01C2qWQWxPP0pjF nNktf1lygVjM1ra2FHVaI8XenwX8T2ICRmWZhw/pO4WuRC+CmQaDGflCQGtk4njA
              jCmYRvWtOeDx/g1bfN43ScvmCmJzSglIyzu7Cuar8nl2a1dYV6eJSODZNKRsSaMy P+e5sbe7ARubTR/N6JylIm4UTnDQ+MGqeksvxTWWGhhN46mLO7hya14WmuhEDjZO
              Zo90HPi1IAT9fcneDRZzH4w7Tb7HKjZ7F0BSu+qa0mvgQ5koubeTooR1LBh9OQ1c oOWGTew3ewQ+GHqwcQFNJ//rgSG6Wcd6LdxqXJjpeY9yuiQR/h71GF6O/r6BHPe7
              vxRLTbgU/5FX/Ck2C2Nu9KL9K3PJhuqaTtwfGkmf8tISvPe2tpRy4tG+6bFFkLYO 9vxdx+8vab1FcGQSDyMpVYCbZOaCfpqfaVJDeH2UYva9cSiX4Pz7HbW3bP4pT4co
              zawiMLq1c/HjFMtf+8YZxYqBu7PLyx2Fg/3F+0NZrdPUXLilLe1Jq5IJqWugMGHK 1BmSPCn9CQ1BYgPRKiDwBQnUMsJYuyHN722H+JSkx8ivpToc9psiJ2mfM/hd3MRY
              t+V6+Vr1KfQXnvrn34m2BF9eoSm0xfSf+H+J+PfOhAWypCbSkE8uQyk2MmCAqJqB wdMYHrfCtA==
          </ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
    </KeyDescriptor>
    ▼<KeyDescriptor use="encryption">
      ▼<ds:KeyInfo>
        ▼<ds:X509Data>
          ▼<ds:X509Certificate>
              MIIF0zCCA7ugAwIBAgIUcbvynCHlIS136kX5xyvff9iNKpQwDQYJKoZIhvcNAQEL BQAweTELMAkGA1UEBhMCVVMxFDASBgNVBAgMC0Nvbm5lY3RpY3V0MRAwDgYDVQQH
              DAdDZW50cmFsMRYwFAYDVQQKDA1BcnRPZl0vc3NpYmxlMRMwEQYDVQQLDApCbG9n V3JpdGVyMRUwEwYDVQQDDAxKZWZmcnlIb3VzZXIwHhcNMTkwNDMwMDIxNjI1WhcN
              MjExMDE2MDIxNjI1WjB5MQswCQYDVQQGEwJVUzEUMBIGA1UECAwLQ29ubmVjdGlj dXQxEDAOBgNVBAcMB0NlbnRyYWwxFjAUBgNVBAoMDUFydE9mUG9zc2libGUxEzAR
              BgNVBAsMCkJsb2dXcml0ZXIxFTATBgNVBAMMDEplZmZyeUhvdXNlcjCCAiIwDQYJ KoZIhvcNAQEBBQADggIPADCCAgoCggIBALNi7Crl/v7PGzo/cMTi6V/bL40Tdsdl
              n7qG96ovRqLnsf5FBsLlWGzd9NpZSssNgcCb4wBkUTTG6re5PmzWiISDWYgr/IYB Ii50fJ+T0HXP5r/WgmTz0dNYjE0LHeq0haIEdEMtZA09bxX+I0kjd3jXBLDxOT14
              87OntMGzRrU5zl0dWKRM2IvtgCiGBaJlE8Vyka09jb7IQySohT0DXoa2xJN28E7l YLPwyVexRT468HbpVJv0Q32vx4YlxqId8vZTD4R3m7jdcXBjJZtPgtYUmN5vtoq+
              SVshNsvW9KvxK+k8O0em0qybXCZ9I8Xw9YNlvGDKUZ6mMESLzFuE9rjxrqPd+eaP kke3VxJk1DtJO8TlBgYC3omo8loQRGe2kMupNQSS8HAiiaZ+S3x0PhxB3L3a9cB4
              AR4t87umNhNDtwlNULMoIuXp9levLiZsm2ep/uVqkkg9ZdY7ZAp+R8xjGwD0cwjx OsqdUBteD4BBi8WPwZbZwGRdbMdHxtyh6sW9qPHiF/WracVPuAG5CpYOp7pWwZXb
              TjIRv9sExTWN6hQKLLEKob5VmWIdUs+AwqfOf3Zex8OO5C4tDpPC77wd0n/y5OCf 7wE85WY96QvkORHCxomoozrANkvYZ1H/AcMmAKxETAaAYoY2tzYLenUFolV544mO
              06AGzbX5rWRRAgMBAAGjUzBRMB0GA1UdDgQWBBRhBJtIimBsy6YKbaNFsoRx+hA6 cjAfBgNVHSMEGDAWgBRhBJtIimBsy6YKbaNFsoRx+hA6cjAPBgNVHRMBAf8EBTAD
              AQH/MA0GCSqGSIb3DQEBCwUAA4ICAQAecYFVgmbHxCrr9TV/D01C2qWQWxPP0pjF nNktf1lygVjM1ra2FHVaI8XenwX8T2ICRmWZhw/pO4WuRC+CmQaDGflCQGtk4njA
              jCmYRvWtOeDx/g1bfN43ScvmCmJzSglIyzu7Cuar8nl2a1dYV6eJSODZNKRsSaMy P+e5sbe7ARubTR/N6JylIm4UTnDQ+MGqeksvxTWWGhhN46mLO7hya14WmuhEDjZO
              Zo90HPi1IAT9fcneDRZzH4w7Tb7HKjZ7F0BSu+qa0mvgQ5koubeTooR1LBh9OQ1c oOWGTew3ewQ+GHqwcQFNJ//rgSG6Wcd6LdxqXJjpeY9yuiQR/h71GF6O/r6BHPe7
              vxRLTbgU/5FX/Ck2C2Nu9KL9K3PJhuqaTtwfGkmf8tISvPe2tpRy4tG+6bFFkLYO 9vxdx+8vab1FcGQSDyMpVYCbZOaCfpqfaVJDeH2UYva9cSiX4Pz7HbW3bP4pT4co
              zawiMLq1c/HjFMtf+8YZxYqBu7PLyx2Fg/3F+0NZrdPUXLilLe1Jq5IJqWugMGHK 1BmSPCn9CQ1BYgPRKiDwBQnUMsJYuyHN722H+JSkx8ivpToc9psiJ2mfM/hd3MRY
              t+V6+Vr1KfQXnvrn34m2BF9eoSm0xfSf+H+J+PfOhAWypCbSkE8uQyk2MmCAqJqB wdMYHrfCtA==
          </ds:X509Certificate>
        </ds:X509Data>
```



```
      <NameIDFormat/>
      <AssertionConsumerService index="1" isDefault="true" Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" Location="http://localhost/login/callback"/>
    </SPSSODescriptor>
  </EntityDescriptor>
```

The great thing about this metadata page is that we can use it to share our internal details with the IdP and the IdP can use it to share its internal details with us. Hopefully we can use it to automate part of our systems so when data changes on one side, the other doesn't have to manually make changes.

Final Thoughts

I know this article makes it sound super easy to set this up, but our team stumbled a bit doing it. My success is because I was able to stand on their shoulders, and I'm happy to share this with you.

For our apps, it is important to secure things up and down the stack and integrating this SSO approach was a big step forward and making that happen.

Nodejs Docker Saml Passportjs Technology