Federating OTRS with mod_auth_mellon

OTRS is the Open-source Ticket Request System, which is a Perl application that runs on an Apache web server. OTRS has two different web interfaces:

- The *customer* interface. This is for people who submit tickets.
- The *agent* interface. This is for people working on tickets ('admins' if you will)

Both the interfaces can use various authentication methods, such as a local database, or Active Directory/LDAP. It is also possible to use external authentication (HTTPBasicAuth) in which case OTRS does not take responsibility for authentication any more, but instead relies on an Apache environment variable to provide the username. The is the way forward if you want to use SAML or federated authentication, but there are some issues with.

The biggest issue is that is not possible to provision accounts in OTRS *before* users have logged in. This is because there is no way of knowing a user's details until they have authenticated. To overcome this I wrote a new customer authentication module for OTRS that creates customer accounts on the fly (auto-provisioning).

At the moment we have no use case yet for auto-provisioning agents. This is left as a future exercise, one idea is to auto-provisioning agents based on the value of a specific SAML attribute.

The standard HTTPBasicAuth can be used for the agent interface.

Below is the recipe for getting OTRS to work with federated authentication using Ubuntu 14.04, OTRS 3.3.8 and mod_auth_mellon 0.7. If you manage to implement it on another combination of software, please let me know.

Prerequisites

Before you start, make sure you have these bits in place:

- A correctly configured Apache web server that is able to serve an HTTPS web site (https://otrs.example.com).
- A SAML Identity Provider (IdP).
- An account on that IdP.
- An attribute that can be used as username in OTRS (for example eduPersonPrincipalName). Attributes for first name, last name, and e-mail are optional but highly recommended as the service would be pretty useless without these. In this case we assume that 'givenName', 'sn', and 'mail' can be used.
- The user name of the to-be administrator account. So, if you choose eduPersonPrincipalName as the attribute for username, you need to know your own value (for instance 'dvisser@surfnet.nl').

OTRS

Go to https://www.otrs.com/try/, scroll to Source, and pick the latest version of OTRS Help Desk.

Follow the instructions at http://otrs.github.io/doc/manual/admin/stable/en/html/index.html, do a standard install and make sure everything works.

Pay special attention to the phrase "Please install OTRS from source, and do not use the OTRS packages that Debian/Ubuntu provides." 🙂

It will require some fiddling to get all the Perl modules sorted, I suggest to use the packages modules as much as possible.

The docs all seem to assume that you'd want to run OTRS inside a subdirectory (https://example.com/otrs), but we want it to be the root of our vhost (https://otrs.example.com), in which case this configuration is a little bit different, see below (you should have the HTTPS stuff already configured, probably in /etc/apache2/mods-enabled/ssl.conf):

```
ServerName otrs.example.com
Alias /otrs-web/ "/opt/otrs/var/httpd/htdocs/"
Alias / "/opt/otrs/bin/cgi-bin/"
<IfModule mod_perl.c>
    # Setup environment and preload modules
    Perlrequire /opt/otrs/scripts/apache2-perl-startup.pl
    # Reload Perl modules when changed on disk
    PerlModule Apache2::Reload
    PerlInitHandler Apache2::Reload
    # mod_perl2 options for GenericInterface
    <Location /nph-genericinterface.pl>
        PerlOptions -ParseHeaders
    </Location>
</IfModule>
<Directory "/opt/otrs/bin/cgi-bin/">
        AllowOverride None
        DirectoryIndex customer.pl
```

```
        AddHandler  perl-script .pl .cgi
        PerlResponseHandler ModPerl::Registry
        Options +ExecCGI
        PerlOptions +ParseHeaders
        PerlOptions +SetupEnv

        # mod_auth_mellon placeholder


    <IfModule mod_version.c>
        <IfVersion < 2.4>
            Order allow,deny
            Allow from all
        </IfVersion>
        <IfVersion >= 2.4>
            Require all granted
        </IfVersion>
    </IfModule>
    <IfModule !mod_version.c>
        Order allow,deny
        Allow from all
    </IfModule>
    <IfModule mod_deflate.c>
        AddOutputFilterByType DEFLATE text/html text/javascript text/css text/xml application/json text/json
    </IfModule>
</Directory>


<Directory "/opt/otrs/var/httpd/htdocs/">
    AllowOverride None
    <IfModule mod_version.c>
        <IfVersion < 2.4>
            Order allow,deny
            Allow from all
        </IfVersion>
        <IfVersion >= 2.4>
            Require all granted
        </IfVersion>
    </IfModule>
    <IfModule !mod_version.c>
        Order allow,deny
        Allow from all
    </IfModule>
    <IfModule mod_deflate.c>
        AddOutputFilterByType DEFLATE text/html text/javascript text/css text/xml application/json text/json
    </IfModule>
    # Make sure CSS and JS files are read as UTF8 by the browsers.
    AddCharset UTF-8 .css
    AddCharset UTF-8 .js
    # Set explicit mime type for woff fonts since it is relatively new and apache may not know about it.
    AddType application/font-woff .woff
</Directory>
<IfModule mod_headers.c>
    # Cache css-cache for 30 days
    <Directory "/opt/otrs/var/httpd/htdocs/skins/*/*/css-cache">
        <FilesMatch "\.(css|CSS)$">
            Header set Cache-Control "max-age=2592000 must-revalidate"
        </FilesMatch>
    </Directory>
    # Cache css thirdparty for 4 hours, including icon fonts
    <Directory "/opt/otrs/var/httpd/htdocs/skins/*/*/css/thirdparty">
        <FilesMatch "\.(css|CSS|woff|svg)$">
            Header set Cache-Control "max-age=14400 must-revalidate"
        </FilesMatch>
    </Directory>
    # Cache js-cache for 30 days
    <Directory "/opt/otrs/var/httpd/htdocs/js/js-cache">
        <FilesMatch "\.(js|JS)$">
            Header set Cache-Control "max-age=2592000 must-revalidate"
        </FilesMatch>
    </Directory>
```

```
    # Cache js thirdparty for 4 hours
    <Directory "/opt/otrs/var/httpd/htdocs/js/thirdparty/">
        <FilesMatch "\.(js|JS)$">
            Header set Cache-Control "max-age=14400 must-revalidate"
        </FilesMatch>
    </Directory>
</IfModule>
```

The site is now configured so that the bare URL will go to the customer interface. This makes the most sense because typically customers will have less clue about where to go.

The agent interface is where you should log in to with the default root@localhost account.

Once you're in, you should create a new agent with full permissions, and make sure the username is your eduPersonPrincipalName.

mod_auth_mellon

mod_auth_mellon is an Apache module, which is available in Ubuntu 14.04 and later. To get this working with Ubuntu 12.04, I recompiled the Debian source packages from the University of Tilburg and made them available in our own APT repository. Either way, it's easy to install:

```
apt-get install libapache2-mod-auth-mellon
a2enmod auth_mellon
```

Create a directory /etc/apache/mellon, and store the Identity Provider metadata in XML format to a file called idp.xml.

Create the cryptographic material for mod_auth_mellon:

```
openssl req -new -newkey rsa:2048 -days 3650 -nodes -x509 -keyout sp.key -out sp.crt
```

Now add this to the configuration of the vhost at the mod_auth_mellon placeholder:

```
        MellonEnable "info"
        MellonSecureCookie On
        MellonSessionDump Off
        MellonUser eduPersonPrincipalName
        MellonSamlResponseDump Off
        MellonEndpointPath "/mellon"
        MellonSPPrivateKeyFile /etc/apache2/mellon/sp.key
        MellonSPCertFile /etc/apache2/mellon/sp.crt
        MellonIdPMetadataFile /etc/apache2/mellon/idp.xml
```

As you can see, the attribute eduPersonPrincipalName is being used as the username. This is the attribute that should **always** be sent by the IdP. In other words, you have to make that authentication doesn't go through when there is an eduPersonPrincipalName. mod_auth_mellon populate the REMOTE_USER environment variable with the value of this SAML attribute.

Keep in mind that eduPersonPrincipalName might work well in a controller environment, such a a national identity federation, where the attribute is always present, validated, etc. It might not be so smart if you use a SAML proxy, in which case you could end up with different users with the same eduPersonPrincipalName.

By this time, you should be able to download the Service Provider metadata from https://otrs.example.com/mellon/metadata, and use it to add it to your IdP, thereby creating a trust relationship.

And once that is done, you should be able to authenticate by going to https://otrs.example.org/mellon.

Adding the new module to OTRS

Create the file Kernel/System/CustomerAuth/HTTPBasicAuthMellon.pm and copy the code below in it:

```
# --
# Kernel/System/CustomerAuth/HTTPBasicAuthMellon.pm
# Provides HTTPBasic authentication for use with Apache's mod_auth_mellon.
# This module auto-provisions customer users.
# Dick Visser <visser@terena.org> 2014-08-22
# Copyright (C) TERENA, http://www.terena.org
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --
package Kernel::System::CustomerAuth::HTTPBasicAuthMellon;
use strict;
use warnings;
sub new {
    my ( $Type, %Param ) = @_;
    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );
    # check needed objects
    for (qw(LogObject ConfigObject DBObject MainObject EncodeObject)) {
        $Self->{$_} = $Param{$_} || die "No $_!";
    }
    $Self->{CustomerUserObject} = Kernel::System::CustomerUser->new( %{$Self} );
    # Mellon environment vars
    $Self->{MailEnvVar}
        = $Self->{ConfigObject}->Get( 'Customer::AuthModule::HTTPBasicAuthMellon::MailEnvVar')
        || 'MELLON_mail';
    $Self->{FirstNameEnvVar}
        = $Self->{ConfigObject}->Get('Customer::AuthModule::HTTPBasicAuthMellon::FirstNameEnvVar')
        || 'MELLON_givenName';
    $Self->{LastNameEnvVar}
        = $Self->{ConfigObject}->Get( 'Customer::AuthModule::HTTPBasicAuthMellon::LastNameEnvVar')
        || 'MELLON_sn';
    $Self->{CustomerIDEnvVar}
        = $Self->{ConfigObject}->Get( 'Customer::AuthModule::HTTPBasicAuthMellon::CustomerIDEnvVar')
        || 'MELLON_customer_id';
    # Debug 0=off 1=on
    $Self->{Debug} = 1;
    $Self->{Count} = $Param{Count} || '';
    return $Self;
}
sub GetOption {
    my ( $Self, %Param ) = @_;
    # check needed stuff
    if ( !$Param{What} ) {
        $Self->{LogObject}->Log( Priority => 'error', Message => "Need What!" );
        return;
    }
    # module options
    my %Option = ( PreAuth => 1, );
    # return option
    return $Option{ $Param{What} };
}
sub Auth {
    my ( $Self, %Param ) = @_;
    # Get attributes values from environment variables
    my $User       = $ENV{REMOTE_USER};
    my $Mail       = $ENV{$Self->{MailEnvVar}} || 'invalid_email@noreply.com';
    my $FirstName  = $ENV{$Self->{FirstNameEnvVar}} || 'first_name';
    my $LastName   = $ENV{$Self->{LastNameEnvVar}} || 'last_name';
```

```
    my $CustomerID = $ENV{$Self->{CustomerIDEnvVar}} || 'default_customer';
    my $RemoteAddr = $ENV{REMOTE_ADDR} || 'Got no REMOTE_ADDR env!';
    # return on no user
    if ( !$User ) {
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message =>
                "No \$ENV{REMOTE_USER}, so not authenticated yet. Redirecting to authenticate (client
REMOTE_ADDR: $RemoteAddr).",
        );
        return;
    }
    # replace parts of login
    my $Replace = $Self->{ConfigObject}->Get(
        'Customer::AuthModule::HTTPBasicAuth::Replace' . $Self->{Count},
    );
    if ($Replace) {
        $User =~ s/^\Q$Replace\E//;
    }
    # regexp on login
    my $ReplaceRegExp = $Self->{ConfigObject}->Get(
        'Customer::AuthModule::HTTPBasicAuth::ReplaceRegExp' . $Self->{Count},
    );
    if ($ReplaceRegExp) {
        $User =~ s/$ReplaceRegExp/$1/;
    }
    # Log Apache environment vars in debug mode
    if ( $Self->{Debug} > 0 ) {
        $Self->{LogObject}->Log(
            Priority => 'debug',
            Message => 'Apache environment vars:'
        );
        foreach my $var (sort keys %ENV) {
            $Self->{LogObject}->Log(
                Priority => 'debug',
                Message =>   $var . "=" . $ENV{$var},
            );
        }
    }
    # log
    $Self->{LogObject}->Log(
        Priority => 'notice',
        Message  => "User '$User' Authentication ok (REMOTE_ADDR: $RemoteAddr).",
    );

    # Auto-provisiong.
    # First check if customer exists
    my %UserTest = $Self->{CustomerUserObject}->CustomerUserDataGet( User => $User );
    if (! %UserTest) {
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message  => "User '$User' doesn't have an account here yet, provisioning it now",
        );
        # Add new customer
        my $newuser = $Self->{CustomerUserObject}->CustomerUserAdd(
            Source         => 'CustomerUser',
            UserFirstname  => $FirstName,
            UserLastname   => $LastName,
            UserCustomerID => $CustomerID,
            UserLogin      => $User,
            UserPassword   => $Self->{CustomerUserObject}->GenerateRandomPassword(),
            UserEmail      => $Mail,
            ValidID        => 1,
            UserID         => 1,
         );
    }
    # return user
    return $User;
}
1;
```

Configuration

When creating a new customer, we also need data for several other fields: first name, last name, e-mail, and customID. Since mod_auth_mellon copies all the SAML attributes into Apache environment variables anyway, we can use them. The module uses the standard SAML attributes (prefixed with MELLON_ because that's their environment variable name) wherever possible:

	Apache environment variable
First name	MELLON_givenName
Last name	MELLON_sn
e-mail address	MELLON_mail
customerID	MELLON_otrs_customer_id

You can override these values in your configuration (`System/Config.pm`) , which at the minimum looks like this:

```
    # Customer Auth
    $Self->{'Customer::AuthModule'} = 'Kernel::System::CustomerAuth::HTTPBasicAuthMellon';
    # Because auto-provisioned users will all have the same e-mail address
    $Self->{CustomerUser}->{CustomerUserEmailUniqCheck} = 0;
    $Self->{'CustomerPanelLoginURL'}    = 'https://otrs.example.com/mellon/login?ReturnTo=/customer.pl';
    $Self->{'CustomerPanelLogoutURL'}   = 'https://otrs.example.com/mellon/logout?ReturnTo=http://www.terena.
org';

    # Uncomment to override the environment vars to be used
    #$Self->{'Customer::AuthModule::HTTPBasicAuthMellon::UsernameEnvVar'} = 'MELLON_eduPersonPrincipalName';
    #$Self->{'Customer::AuthModule::HTTPBasicAuthMellon::MailEnvVar'} = 'MELLON_mail';
    #$Self->{'Customer::AuthModule::HTTPBasicAuthMellon::FirstNameEnvVar'} = 'MELLON_givenName';
    #$Self->{'Customer::AuthModule::HTTPBasicAuthMellon::LastNameEnvVar'} = 'MELLON_sn';
    #$Self->{'Customer::AuthModule::HTTPBasicAuthMellon::CustomerIDEnvVar'} = 'MELLON_otrs_customer_id';


    # Agents are NOT auto-provisioned. The will have to be created manually.
    # To find their username, they could first log in as a customer, so that you can see their username
    # in the Customer User Manager overview.
    $Self->{'AuthModule'} = 'Kernel::System::Auth::HTTPBasicAuth';
    $Self->{'LoginURL'} = 'https://otrs.example.com/mellon/login?ReturnTo=/index.pl';
    $Self->{'LogoutURL'}        = 'https://otrs.example.com/mellon/logout?ReturnTo=http://www.terena.org';
```

I just found out that this set-up won't work unless you put the configuration file ZZZAAuth.pm in `Kernel/Config/Files`, and give it permissions so that otrs/www-data has write access .There are many statements in there, but I do not yet know which ones are needed... To be continued.

At this point, you should be able to log in to the site as an admin with your new account.

If you log in to the customer page, your account will be automatically created.

I don't even trust my own Perl skills, so use all of this with care 🙂

Known limitations

Customers cannot edit their details

Once customers are auto-provisioned, they cannot edit their name, e-mail, or any other values. This might be an issue in your environment. Only agents can edit this.

Unknown agents get stuck in an authentication loop

When agents that don't have an account try to log in, they get stuck in an endless authentication loop. Another agent should first create a new agent account.

If the username isn't know yet, then the new agent could first log in to the customer interface. The existing agent can then see what username was used, and use this to create an agent account.