Setting up U2F Multi-factor authentication with Shibboleth IdP for use within SWAMID

> ⚠ **Work in progress**
>
> SWAMID Operations are still working with this documentation.

This document is a suggested method for Multi-Factor Authentication using Shibboleth and Yubico's u2fval validation server. Supported hardware includes Yubikey Security key, Yubikey 4 and 5 (https://www.yubico.com/products/yubikey-hardware/), as well as U2F tokens from other manufacturers.

U2F works with Google Chrome, Firefox and the newest version of Microsoft Edge (based on Chromium).

This tutorial will cover:

1. Installation of Yubico's u2fval server and configuring it to be used with a Shibboleth Identity Provider
2. Installation of a prototype web connector to ease registration of Yubikeys for use with the IdP
3. Installation and configuration of MFA and U2F modules on a Shibboleth IdP

Requirements:

1. You are familiar with Linux. The examples below are tested (once!) with Centos 7.
2. You have an existing test IdP which you can use to test MFA without affecting your production IdP.

Getting started:

If you intend to follow the whole tutorial and also install on real servers (as opposed to running stuff as localhost on your own Linux workstation), then you need to install two servers. One will be the u2fval server (in our examples called u2fval.example.se) and the other a prototype connector server for administering the u2fval server (in our examples called register.example.se).

The examples below use Centos, but any Linux will do. You may, however, need to adjust some of the syntax and config code for other distributions.

Useful extras:

Ansible roles and playbook file are available for the u2fval server for those running Ansible 1.9.6. They may or may not work for later versions of Ansible. They can be downloaded here.

Installing the u2fval server

The u2fval validation server is used to store the U2F data for your users. We are going to use Yubico's u2fval software, https://developers.yubico.com/u2fval/.

This install takes place on the u2fval.example.se you previously installed. You can try to use the provided Ansible roles package if you have an existing Ansible infrastructure. Otherwise the steps you need to take are outlined here and complement the information on Yubico's website (https://developers.yubico.com/u2fval/). We suggest you read them in tandem. Even if you don't use Ansible, you may find it useful to look at the example files as they contain working templates.

Server prerequisites

Install the Centos SCL (Software collections):

```
yum install centos-release-scl
```

Install Python 3.6 with wsgi support, Apache 24 (from SCL) and mod_ssl

```
yum install rh-python36 rh-python36_wsgi httpd24 httpd24-mod_ssl
```

Ensure that Python 3.6 is the default, by putting the following file (suggested name: enablepython36.sh) in /etc/profile.d:

Reboot your server.

Install u2fval (https://developers.yubico.com/u2fval/ and https://developers.yubico.com/u2fval/Installation.html)

Create a folder for the u2fval package to reside. In the Ansible example and the remainder of this document, this is assumed to be /opt/u2fval. The folder should be rwx by the apache group (at least).

```
cd /opt/u2fval
virtualenv venv
source venv/bin/activate
pip install -U pip
pip install u2fval
```

Create a folder for the u2fval config file and the subfolder metadata. It must reside under /etc/yubico/u2fval. Create the default u2fval.conf file containing the following:

```
# U2FVAL settings file

# For testing/debugging only, set to False in production!
DEBUG = True
TESTING = True

# Set to False to disable pretty-printing of JSON responses.
# Note: XMLHttpRequests are never pretty-printed.
JSONIFY_PRETTYPRINT_REGULAR = True

# Database configuration
SQLALCHEMY_DATABASE_URI = 'sqlite:////etc/yubico/u2fval/u2fval.db'
SQLALCHEMY_TRACK_MODIFICATIONS = False

# If True, use memcached for storing registration and authentication requests
# in progress, instead of persisting them to the database.
USE_MEMCACHED = False

# If memcached is enabled, use these servers.
MEMCACHED_SERVERS = ['127.0.0.1:11211']

# Add files containing trusted metadata JSON to the directory below.
METADATA = '/etc/yubico/u2fval/metadata/'

# Allow the use of untrusted (for which attestation cannot be verified using
# the available trusted metadata) U2F devices.
ALLOW_UNTRUSTED = False
```

As you can see, we are not going to complicate matters by using memcache or anything other than a sqlite database file at present. You can fix that yourself later once you've got things working.

Initialize the sqlite database

```
u2fval db init
```

To be able to use the server, a client needs to be created. This is done using the `u2fval client create`command. Look at the help for this command (https://developers.yubico.com/u2fval/Manuals/u2fval.1.html). We are going to create a Multi-faceted app as described at: https://developers.yubico.com /U2F/App_ID.html. This allows both your IdP and registration server to access the AppID using the same digest credentials (which we will create below).

Create a file which you need put on a web origin server (e.g. www.kau.se or www.su.se etc) containing the following JSON (adjusted to include the actual hostnames relevant for your installation). The recommended filename is app-id.json. See https://www.kau.se/app-id.json for a real world example.

```
{
  "trustedFacets" : [{
    "version": { "major": 1, "minor" : 0 },
    "ids": [
        "https://register.example.se",
        "https://idp-test.example.se"
    ]
  }]
}
```

Create the client using the u2fval command line app:

```
u2fval client create CLIENTNAME https://www.example.se/app-id.json https://register.example.se https://idp-test.
example.se
```

Replace the name CLIENTNAME with a suitable name (e.g. your University's name), the AppID with the URL to your app-id.json file and the FacetIDs with the real URLs to your registration service (which we cover later) and your test IdP. All these URLs MUST be protected by SSL and must not contain any path or trailing slash.

Apache deployment (https://developers.yubico.com/u2fval/Apache_Deployment.html)

Create a server certificate for your Apache webserver and have the private key and certificate with associated chain installed in the /etc/pki/tls/private and /etc/pki/tls/certs directory. How to do this is outside the scope of this document.

You will need to configure your Apache HTTP server. An example http.conf is included in the Ansible package in the file template/u2fval.example.se-httpd. conf.j2 file. Alternatively see the example at https://developers.yubico.com/u2fval/Apache_Deployment.html. You'll need to add the following virtual hosts to the http.conf file. Note the location of this file is /opt/rh/httpd24/root/etc/httpd/conf. Remember to change the hostnames below (and the paths to keys and certificates) to the correct values for your installation.

Make sure mod_auth_digest and mod_wsgi (modules/mod_rh-python36-wsgi.so) are enabled.

```
<VirtualHost *:80>
 ServerName u2fval.example.se
 Redirect temp "/" "https://u2fval.example.se/"
</VirtualHost>

<VirtualHost *:443>
 ServerName u2fval.example.se
 DocumentRoot /opt/rh/httpd24/root/var/www/html

 <IfModule mod_wsgi.c>
    WSGIDaemonProcess u2fval python-home=/opt/u2fval/venv
    WSGIApplicationGroup %{GLOBAL}
    WSGIScriptAlias /wsapi/u2fval /opt/u2fval/u2fval.wsgi process-group=u2fval

    <Directory /opt/u2fval/>
        Options None
        AllowOverride None
        AuthType Digest
        AuthName "u2fval"
        AuthUserFile /opt/u2fval/clients.htdigest
        Require valid-user
    </Directory>

  </IfModule>

  SSLEngine On
  SSLCertificateFile /etc/pki/tls/certs/server.crt
  SSLCertificateKeyFile /etc/pki/tls/private/private.key
  SSLCACertificateFile /etc/pki/tls/certs/DigiCertCA.crt
  ...plus other ssl config...
</VirtualHost>
```

Create the u2fval.wscgi file under /opt/u2fval. This is the script alias for u2fval in the Apache VirtualHost above.

```
from u2fval import app
import logging
from logging.handlers import RotatingFileHandler

# First we remove the default logging handlers.
for handler in app.logger.handlers:
    app.logger.removeHandler(handler)
#
# Now we add our own.
handler = RotatingFileHandler('/var/log/u2fval.log', maxBytes=100000, backupCount=1)
handler.setLevel(logging.DEBUG)
formatter = logging.Formatter('[%(levelname)s] %(asctime)s %(name)s: %(message)s', '%Y-%m-%d %I:%M:%S')
handler.setFormatter(formatter)
app.logger.addHandler(handler)

application = app
```

The above http.conf configuration points out an AuthUserFile which does not yet exist. This is where client credentials will be stored, so let's create the file and add our client now:

```
htdigest -c /opt/u2fval/clients.htdigest "u2fval" CLIENTNAME
```

where CLIENTNAME matches the clientname you used above in the u2fval command above. Enter a password twice and make a note of it. You'll need it later for both your registration server and test IdP.

Start Apache and check its logs for any errors.

Prototype web connector

Yubico provides a couple of example connector libraries (https://developers.yubico.com/Software_Projects/FIDO_U2F/U2FVAL_Connector_Libraries/) to help you get started with using your u2fval server. SWAMID Operations has tested the PHP client and despite a few problems with the example code has managed to get it working. The connector can be found at https://github.com/Yubico/u2fval-client-php

Look in the examples/tutorial directory. It may look empty, but there are a number of separate tags which contain the example connector code.

You'll need to install the connector on the webserver that we referred to as register.example.se above. We recommend Apache with SSL support, rh-php71 from the software collections configured with php-fpm. Installing and configuring these is outside of the scope of this document. If you wish to adopt a similar approach, then you should aim to configure relevant parts of your http.conf as below to serve the correct document root:

```
<VirtualHost *:80>
   ServerName register.example.se
   Redirect temp "/" "https://register.example.se/"
</VirtualHost>

<VirtualHost *:443>
   ServerName register.example.se
   # In the directory below, we have created a PHP-FPM pool called register and that pool is served from /var/www
/register
   DocumentRoot /var/www/register/u2fval-client-php/examples/tutorial

   <FilesMatch \.php$>
       SetHandler "proxy:fcgi://127.0.0.1:9001"
    </FilesMatch>

   SSLEngine on
   ....

</VirtualHost>
```

The most important part to note above is that the document root needs to be in the tutorial directory and not the u2fval-client-php directory. This httpd.conf config is for inspiration only!

Clone the repo into the /var/www/register directory:

```
git clone https://github.com/Yubico/u2fval-client-php.git
```

Install the dependencies:

```
composer.phar install
```

and then switch the tutorial directory to the correct tag (in initial testing, tag tutorial-5 doesn't seem to work and causes internal server errors)

```
cd u2fval-client-php/examples/tutorial
git checkout tutorial-4
```

You need to update the u2f-api.js which is stored in the js directory in the tutorial. The version packaged is out-of-date and doesn't work. Download a new version from https://demo.yubico.com/js/u2f-api.js

You need to update the config.php file stored in the examples directory to point to your u2fval server. On that server, we used HTTP digest auth, so uncomment and update that line as follows, where clientname and password are those you provided to the htdigest and u2fval commands above.

```
$u2fval = U2fVal\Client::withHttpAuth('https://u2fval.example.se/wsapi/u2fval', 'CLIENTNAME', 'PASSWORD');
```

Start your webserver. If you can see the "Hello, U2F!" page on your website, then you are ready to test.

Try to see if you can register a Yubikey for one of the three inbuilt test users.

If that works, then change one of the usernames in the $users array located in the users.inc.php file to a real username that you can authenticate with on your Identity provider later.

Configuring your test IdP for MFA

You must be running Shibboleth Identity Provider v3.3.x. If not, upgrade first: SWAMID Webinar 2 2017 - Uppgradera Shibboelth Identity Provider till senaste versionen

The updates to your test IdP involve configuring MultiFactorAuthnConfiguration (https://wiki.shibboleth.net/confluence/display/IDP30/MultiFactorAuthnConfiguration) and adding in an extra module for u2f which is available on Github at https://github.com/Ratler/shibboleth-mfa-u2f-auth.

If you do not have the conf/authn/mfa-authn-config.xml in your Shibboleth installation directory, then you can copy it from the /opt/shibboleth-identity-provider-3.3.2/conf/authn distribution directory.

Configure your idp.properties so that MFA is the only enabled flow:

```
idp.authn.flows= MFA
```

Add the u2f.properties file to the comma-delimited list of property resources in the idp.properties file, for example:

```
idp.additionalProperties= /conf/ldap.properties, /conf/saml-nameid.properties, /conf/services.properties, /conf
/u2f.properties
```

Your u2f.properties file should be configured as follows:

```
####################
# General settings #
####################

## U2F Application Facets - see https://developers.yubico.com/U2F/App_ID.html) for more information
u2f.appId=https://idp-test.example.se

## Data store configuration - available data stores are, dummyDataStore and yubicoU2fValidationDataStore
## Uncomment one of the data stores and it's configuration properties, update the empty properties where needed.

#################################
# dummyDataStore (testing only) #
#################################
# u2f.dataStore=dummyDataStore
# u2f.dummyStore.keyHandle=
# u2f.dummyStore.public=

################################
# yubicoU2fValidationDataStore #
################################
u2f.dataStore=yubicoU2fValidationServerDataStore
## URL to the U2Fval API
u2f.u2fval.endPoint=https://u2fval.example.se/wsapi/u2fval
## U2Fval HTTP Basic Authentication set the properties u2f.u2fval.username and u2f.u2fval.password
u2f.u2fval.username=CLIENTNAME
u2f.u2fval.password=PASSWORD
## For HTTP Digest Authentication also set the property u2f.u2fval.realm
u2f.u2fval.realm=u2fval
```

Again, replace CLIENTNAME and PASSWORD with those you set at the htdigest and u2fval commands above. The appId MUST be the URL to your test IdP without any path information or trailing slash. Add this

The following must be added to the conf/authn/general-authn.xml so that both authn/MFA and authn/U2f are available:

```
<bean id="authn/U2f" parent="shibboleth.AuthenticationFlow" p:passiveAuthenticationSupported="true" p:
forcedAuthenticationSupported="true">
   <property name="supportedPrincipals">
     <util:list>
       <bean parent="shibboleth.SAML2AuthnContextClassRef" c:classRef="https://refeds.org/profile/mfa" />
       <bean parent="shibboleth.SAML1AuthenticationMethod" c:method="https://refeds.org/profile/mfa" />
     </util:list>
   </property>
</bean>

<bean id="authn/MFA" parent="shibboleth.AuthenticationFlow" p:passiveAuthenticationSupported="true" p:
forcedAuthenticationSupported="true">
   <property name="supportedPrincipals">
     <list>
       <bean parent="shibboleth.SAML2AuthnContextClassRef" c:classRef="https://refeds.org/profile/mfa" />
       <bean parent="shibboleth.SAML1AuthenticationMethod" c:method="https://refeds.org/profile/mfa" />
       <bean parent="shibboleth.SAML2AuthnContextClassRef" c:classRef="urn:oasis:names:tc:SAML:2.0:ac:classes:
PasswordProtectedTransport" />
       <bean parent="shibboleth.SAML2AuthnContextClassRef" c:classRef="urn:oasis:names:tc:SAML:2.0:ac:classes:
Password" />
       <bean parent="shibboleth.SAML1AuthenticationMethod" c:method="urn:oasis:names:tc:SAML:1.0:am:password" />
     </list>
   </property>
 </bean>
```

The conf/authn/mfa-authn-config.xml must define the transition maps that control the flow from one factor to another. The following example defines password as the first factor followed by U2f if the service provider demands it using the authentication context of https://refeds.org/profile/mfa:

```
<util:map id="shibboleth.authn.MFA.TransitionMap">
   <!-- First rule runs the Password login flow. -->
   <entry key="">
     <bean parent="shibboleth.authn.MFA.Transition" p:nextFlow="authn/Password" />
   </entry>

  <!-- Second rule runs a function if Password succeeds, to determine whether an additional factor is required. --
>
   <entry key="authn/Password">
     <bean parent="shibboleth.authn.MFA.Transition" p:nextFlowStrategy-ref="checkSecondFactor" />
   </entry>

  <!-- An implicit final rule will return whatever the final flow returns. -->
</util:map>

<!-- Example script to see if second factor is required. -->
<bean id="checkSecondFactor" parent="shibboleth.ContextFunctions.Scripted" factory-method="inlineScript" p:
customObject-ref="shibboleth.AttributeResolverService">
  <constructor-arg>
    <value>
    <![CDATA[
      nextFlow = "authn/U2f";

      authCtx = input.getSubcontext("net.shibboleth.idp.authn.context.AuthenticationContext");
      mfaCtx = authCtx.getSubcontext("net.shibboleth.idp.authn.context.MultiFactorAuthenticationContext");

      if (mfaCtx.isAcceptable()) {
          nextFlow=null;
      }

      nextFlow; // pass control to second factor or end with the first
    ]]>
    </value>
  </constructor-arg>
</bean>
```

Restart and test. Testing via https://sp.swamid.se should result in only being asked for your password. Testing via https://mfa-check.swamid.se should result in both forceAuth and usage of the second factor.