

---

# Access control in Apache with the OpenSSO PHP extension and Auth MemCookie

Olav Morken

Mon Jul 16 08:34:22 2007

## Table of Contents

Introduction .....	1
Prerequisites .....	1
Apache 2 .....	1
PHP .....	2
dev.java.net account .....	2
CVS client .....	2
gcc .....	2
General layout .....	2
memcache .....	3
memcached .....	3
libmemcache .....	3
memcache PHP extension .....	4
The OpenSSO PHP Extension .....	4
Installing the OpenSSO PHP extension .....	4
Configuring the OpenSSO PHP extension .....	4
Service provider metadata file for IdP .....	8
Auth MemCookie .....	9
Installing Auth MemCookie .....	9
Configuring Auth MemCookie .....	10
Testing the final result .....	11
Logging out .....	12

## Introduction

This HOWTO describes how to add access control to a web site by using the OpenSSO PHP extension and Auth MemCookie.

The OpenSSO PHP extension is used to authenticate the user against a SAML 2.0 single signon server and Auth MemCookie is used for access control after the user is authenticated.

## Prerequisites

### Apache 2

Since Auth MemCookie is an Apache 2 module, the web site you are going to add access control to must be hosted on an Apache 2 web server.

To protect personal data, which will be sent from the IdP to your web site through a standard HTTP POST request, you should enable SSL on your web site. How to configure SSL for your web site is outside of the scope of this document.

Since we are going to compile Auth MemCookie from source, we are going to need development headers and tools for Apache 2. In Debian Etch, these can be installed by running:

```
apt-get install apache2-prefork-dev
```

Other distributions may include the development tools and headers in the same package as the web server, or they may be in a separate package.

You can verify that you have the required development tools by running:

```
apxs2 -q TARGET
```

This should print out apache2. If apxs2 fails to run, then you can try to replace apxs2 with apxs.

## PHP

The OpenSSO PHP extension requires a PHP enabled web server which supports OpenSSL.

On Debian Etch you can install PHP by running:

```
apt-get install php5
```

## dev.java.net account

Currently the source code for the OpenSSO PHP extension is only available through CVS. To access the CVS repository at dev.java.net you must have an account there. You can get a dev.java.net account by registering at <https://www.dev.java.net/servlets/Join>.

## CVS client

To download the source code for the OpenSSO PHP extension you need a CVS client. On Debian Etch, you can install the standard CVS client by running:

```
apt-get install cvs
```

## gcc

To compile Auth MemCookie you are going to need gcc. On Debian Etch, you can install gcc by running:

```
apt-get install gcc
```

## General layout

In this document we assume that your web site is a service provider located at the `sp.example.com` domain. We assume that the area you want to add access control to is located under `https://sp.example.com/protected/`. We are going to install the OpenSSO PHP extension into

the `/openssophp/` directory on the web server. Through this document we are going to assume that the root directory of your web site is located at `/var/www/`.

## memcache

memcache is a system for storing key-value pairs in a memory database. The system consists of memcached - a server which stores data, and a number of client APIs. We are going to use libmemcache - a C API, and the memcache PHP extension, which allows PHP scripts to access memcached servers.

## memcached

memcached (<http://danga.com/memcached/>) is used to store data in memory and share this data between multiple processes. Here we are going to use it to store authentication data and share this data between multiple Apache processes.

## Installing memcached

memcached is in widespread use, and you can most likely install it through your distributions package management interface.

In the case of Debian Etch you can install memcached by running:

```
apt-get install memcached
```

This will install the server and start it.

## Configuring memcached

How the configuration of memcached is stored depends on the distribution. In Debian Etch the configuration is stored in `/etc/memcached.conf`.

memcached has no authentication, and it is therefore important that you limit the ability to connect to the memcache server to trusted computers. If memcached runs on the same computer as the webserver, then you can limit the connections to be from the webserver by configuring memcached to bind to the loopback-address of the computer. To do this you should add the following line to the configuration file:

```
-l 127.0.0.1
```

If you want to run memcached on a different computer than the webserver, you may have to block connections to memcached by using a firewall. Configuration of a firewall is outside of the scope of this HOWTO.

## libmemcache

libmemcache is a library which is used to access memcache servers from other applications. libmemcache will be used by Auth MemCookie.

On Debian Etch you can install libmemcache and the development header files by running:

```
apt-get install libmemcache0 libmemcache-dev
```

This library is most likely also available in other distributions. Search your distributions package repository for libmemcache.

If it is not available, then you can download the source code from this web page: <http://people.freebsd.org/~seanc/libmemcache/>

## memcahce PHP extension

This is a PHP extension which allows PHP scripts to access memcached servers. It is required by the authmemcookie OpenSSO PHP plugin.

On Debian Etch, you can install this PHP extension by running:

```
apt-get install php5-memcache
```

It may also be available under a similar name on other distributions. If it is not available in your distribution, then you will have to install it by yourself.

This extension is part of the PECL repository of PHP extensions. The homepage for this extension can be found at: <http://pecl.php.net/package/memcache>

For instructions on how to install PECL PHP extensions, please refer to: <http://www.php.net/manual/en/install.pecl.php>

## The OpenSSO PHP Extension

The OpenSSO PHP extension is a set of php files which are used to communicate with a SAML 2.0 Identity Provider.

### Note

This section slightly overlaps the *OpenSSO PHP Extension User's Manual*. Consequently crosschecking with the User's Manual is a good idea if things are unclear. In the future parts of this section will be merged into the User's Manual.

## Installing the OpenSSO PHP extension

The OpenSSO PHP extension can be checked out of a CVS repository at [dev.java.net](http://dev.java.net) [<http://dev.java.net>]. First you need a username and password on [dev.java.net](http://dev.java.net) [<http://dev.java.net>]. Then you can use the following commands to check out the source code for the OpenSSO PHP extension:

```
cvs -d :pserver:USERNAME@cvs.dev.java.net:/cvs login
cvs -d :pserver:USERNAME@cvs.dev.java.net:/cvs checkout \
    opensso/extensions/saml2php/openssophp
```

Replace USERNAME with your username on [dev.java.net](http://dev.java.net), and enter your password when asked for it.

Afterwards you can find the OpenSSO PHP extension under `opensso/extensions/saml2php/openssophp` in your current working directory. Copy `opensso/extensions/saml2php/openssophp` to a directory on your web site:

```
cp -rv opensso/extensions/saml2php/openssophp /var/www
```

You should change `/var/www` if the root of your web site is not located at `/var/www` or if you don't want the OpenSSO PHP extension to be located under `/openssophp` on your web site.

## Configuring the OpenSSO PHP extension

There are three template files in `/var/www/openssophp/config/`: `config.php.template`, `saml-metadata-IdP.php.template` and `saml-metadata-SP.php.template`. We are going to use these as a starting point for our configuration. First we copy the files and give them their proper name:

```
cd /var/www/openssophp/config
cp config.php.template config.php
cp saml-metadata-IdP.php.template saml-metadata-IdP.php
cp saml-metadata-SP.php.template saml-metadata-SP.php
```

## config.php

Open `config.php` in your favourite text editor. We need to update the following options: `basedir`, `baseurl`, `spi-sessionhandling`, `spi-namemapping` and `defaultLandingPage`.

`basedir` should be set to the full path to the `openssophp` directory. In this example this would be `/var/www/openssophp`.

`baseurl` is the full URL to the `openssophp` directory. This is `https://sp.example.com/openssophp/` in this example.

`spi-sessionhandling` determines which plugin the OpenSSO PHP extension should use for session handling. This should be set to `authmemcookie`. `authmemcookie` is a session handler which stores the session state in a form which is compatible with the requirements of Auth MemCookie.

`spi-namemapping` selects which plugin the OpenSSO PHP extension should use to map NameIDs from the IdP server to local users. In this example we will set this to `transient`.

`defaultLandingPage` is the URL which the user should see after he is authenticated. In this example we will set it to `https://sp.example.com/protected/`, which is the root of our protected area on the web site.

The `authmemcookie` section contains reasonable defaults if you run the memcache server on the same computer as the webserver. If you run the memcache server on a different computer, then you will have to update the `memcache_servers` option.

The `userdatabase` section is ignored unless you use the database `namemapping` plugin.

The final result should look something like this:

```
$LIGHTBULB_CONFIG = array (
    'basedir'          => '/var/www/openssophp/',
    'baseurl'          => 'https://sp.example.com/openssophp/',

    'spi-sessionhandling' => 'authmemcookie',
    'spi-namemapping'    => 'transient',

    /* Configuration for the authmemcookie session handler. */
    'authmemcookie'     => array (
        /* The list of memcache servers. This is a string of
         * host:port-pairs, separated by ','. The port number is
         * optional if the server runs on the default port (11211).
         * Example: 'localhost,remote_a:22122,remote_b'
         */
        'memcache_servers' => '127.0.0.1:11211',

        /* The name of the cookie. Should match the
         * Auth_memCookie_CookieName configuration parameter for
         * Auth MemCookie. The default value for
         * Auth_memCookie_CookieName is 'AuthMemCookie'.
```

```
    */
    'cookie_name'          => 'AuthMemCookie',

    /* The path the cookie is valid under.
     * This path must include both the OpenSSO PHP installation, and
     * the pages protected by Auth MemCookie.
     */
    'cookie_path'          => '/',

    /* The domain the cookie is valid for. Use an empty string for
     * the default value.
     */
    'cookie_domain'        => '',

),

'userdatabase'            => array (
    'username'             => 'openssodemo',
    'password'              => '8s732j',
    'host'                  => 'localhost',
    'database'              => 'openssodemo'
),

'defaultLandingPage'      => 'https://sp.example.com/protected/'
);
```

## saml-metadata-IdP.php

This is the file where you add configuration for your IdP. The IdP is the identity provider for your service.

### Note

Note that this file can contain configuration for multiple IdPs. Every IdP is identified by an entity ID.

When making a login request, the OpenSSO PHP extension will use the first IdP in the list by default. When receiving a reply to a login request the OpenSSO PHP extension will look at the issuer of the reply to determine which IdP to validate the reply against. It is therefore important that you remove any untrusted IdPs, from the configuration.

Every IdP has three options: `SingleSignOnUrl`, `SingleLogoutUrl` and `certFingerprint`. These values should be supplied by your IdP.

If you received an XML file describing your IdP, then you can find the entity ID of the IdP and the values for `SingleSignOnUrl` and `SingleLogoutUrl` in this file.

To find the entity ID, look at the first XML element in the file. It should look like this:

```
<EntityDescriptor entityID="idp.example.com"
  xmlns="urn:oasis:names:tc:SAML:2.0:metadata">
```

You can find the entity ID in the `entityID` attribute of this element.

To find the value for the `SingleSignOnUrl` option you should look for an XML element similar to the following:

```
<SingleSignOnService
  Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
  Location="https://idp.example.com/amserver/SSORedirect/metaAlias/idp"
/>
```

There may be several similar elements with different values in the Binding attribute. We want the element where the binding ends with HTTP-Redirect.

This element gives the value for the SingleSignOnUrl in the Location attribute.

For the value of the SingleLogoutUrl option, you should find an XML element similar to the following:

```
<SingleLogoutService
  Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
  Location="https://idp.example.com/amserver/IDPSloRedirect/metaAlias/idp"
  ResponseLocation=
    "https://idp.example.com/amserver/IDPSloRedirect/metaAlias/idp"
/>
```

Again, there may be several similar elements, but we want the one where the Binding attribute ends with HTTP-Redirect.

The correct value for the SingleLogoutURL option is given by the Location attribute.

certFingerprint is the fingerprint of the IdP's public key. If you received the IdP's public key in the form of a pem-file, then you can use the following command to get the fingerprint:

```
openssl x509 -in idp-public-key.pem -fingerprint
```

Replace idp-public-key.pem with the filename of the public key. The fingerprint will be in the first line of output. This line will be similar to the following:

```
SHA1 Fingerprint=<fingerprint>
```

Just copy the text after the equal sign and insert it into the configuration file as the value for certFingerprint.

The result should look similar to this:

```
$idpMetadata = array (
  "idp.example.com" => array (
    "SingleSignOnUrl" =>
      "https://idp.example.com/amserver/SSORedirect/metaAlias/idp",
    "SingleLogoutUrl" =>
      "https://idp.example.com/amserver/IDPSloRedirect/metaAlias/idp",
    "certFingerprint" =>
      "6c:ad:e0:5c:e0:0e:12:b2:93:dd:94:04:33:e3:2e:4c:8e:c4:e5:65"
  )
);
```

## saml-metadata-SP.php

This file contains the description of your service provider. The service provider is the web site you want to add access control to.

You can list several service providers in this file. The OpenSSO PHP extension will use the first one by default.

There are three parameters which should be set: assertionConsumerServiceURL, issuer and spNameQualifier.

`assertionConsumerServiceURL` is the URL of `AssertionConsumerService.php` in the `openssophp/` directory. In our case this will be `https://sp.example.com/openssophp/AssertionConsumerService.php`.

`issuer` is the entity ID of the service provider. This is typically the hostname of the web site, and would be set to `sp.example.com` in our example.

`spNameQualifier` is used to affiliate multiple service providers. This type of configuration is outside of the scope of this HOWTO. It is recommended to set this option to be equal to the `issuer` option (`sp.example.com` in our case).

The final result will look like this:

```
$spMetadata = array (
    "/sp" => array(
        "assertionConsumerServiceURL" =>
            "https://sp.example.com/openssophp/AssertionConsumerService.php",
        "issuer" => "sp.example.com",
        "spNameQualifier" => "sp.example.com"
    )
);
```

`/sp` is the identifier for this service provider, and can be used by the OpenSSO PHP extension to differentiate between different service providers.

## Service provider metadata file for IdP

Next, we need to configure your IdP to work together with the OpenSSO PHP extension installed on your web site. Your IdP will most likely require a XML file which describes the configuration on your site.

The following text is a template which you may adapt to match your web site:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<EntityDescriptor
    entityID="sp.example.com"
    xmlns="urn:oasis:names:tc:SAML:2.0:metadata" >

    <SPSSODescriptor
        AuthnRequestsSigned="false"
        WantAssertionsSigned="false"
        protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol" >

        <SingleLogoutService
            Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
            Location="https://sp.example.com/openssophp/LogoutListener.php" />

        <NameIDFormat>
            urn:oasis:names:tc:SAML:2.0:nameid-format:transient
        </NameIDFormat>

        <AssertionConsumerService
            index="0"
            isDefault="true"
            Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
            Location=
                "https://sp.example.com/openssophp/AssertionConsumerService.php"
            />

    </SPSSODescriptor>
```



```
</EntityDescriptor>
```

You should change all references to `sp.example.com` to the domain of your web site. The value of the `entityID` attribute specified in the `EntityDescriptor` element must match the value you added as `issuer` in `saml-metadata-SP.php`.

The value of the `Location` attribute in the `AssertionConsumerService` element should match the value of the `assertionConsumerServiceURL` option in `saml-metadata-SP.php`.

This file should be added to your IdP. How this is done is outside of the scope of this HOWTO.

## Auth MemCookie

Auth MemCookie is an Apache 2 module for access control. This module uses data stored on a memcache server to authenticate a user. The key to the authentication data is stored in a cookie (by default named `AuthMemCookie`).

Auth MemCookie only deals with access control. Authentication of the user must be handled by another system. In our case this other system will be the OpenSSO PHP extension.

When authenticating a request Auth MemCookie walks through the following steps:

1. Get the session id. The session id is stored in a cookie (by default named `AuthMemCookie`).
2. Get the session data. Auth MemCookie fetches session data by looking up the session id on the memcache server.
3. Verify the remote ip. Auth MemCookie checks the ip address stored in the session data against the ip address of the current request. This step is optional, and can be disabled by setting the `AuthMemCookie_MatchIP` option to `no`.
4. Get username and groups from session data. The username is stored in the `UserName` field in the session data and the groups the user is a member of is stored in the `Groups` field.
5. Check username and groups against `Require` configuration directives. See <http://httpd.apache.org/docs/2.0/mod/core.html#require>

If any of the steps 1-4 fails, then Auth MemCookie will return a `401 Authorization Required` error. A `403 Forbidden` error will be returned if the last step fails.

When a user is successfully authenticated, Auth MemCookie will store all the fields from the session data in environment variables accessible to the web page. Every field will be stored by setting `MCAC_<field-name>` to the value of the field. See section 7 for an example of how to access these fields from a PHP script.

## Installing Auth MemCookie

Auth Memcookie can be downloaded from: <http://authmemcookie.sourceforge.net/>

First you need to extract the files in the source archive:

```
tar xvzf mod_authmemcookie_v1.0.1.tar.gz
```

The version of Auth MemCookie may have changed since this HOWTO was written, and it is possible

that the exact filenames has changed.

Next, you need to edit the file named `Makefile`. In the beginning of the file you will find a line similar to the following:

```
MY_APXS=/product/apache/moteur/2.0.52/bin/apxs
```

You need to replace `/product/apache/moteur/2.0.52/bin/apxs` with the path to the `apxs` executable. On Debian Etch you can replace it with just `apxs2`, but, depending on your distribution, you may have to specify the full path. The executable may be named `apxs` or `apxs2`.

### Note

Note that `apxs` is part of the Apache distribution. You may be necessary to install developement headers for Apache 2 to get the executable. On Debian Etch you can install the package `apache2-prefork-dev`.

After updating the `makefile`, you can compile and install Auth MemCookie by running the following commands:

```
make
make install
```

## Configuring Auth MemCookie

Now we need to change the Apache 2 configuration to use Auth MemCookie for access control.

The precise location of the configuration is distribution specific, but in the general case, you should add everything in `/etc/apache2/httpd.conf`, or a similar path. In Debian Etch the configuration is spread into several files stored in `/etc/apache2/mods_available/`.

We are going to add two files in that directory. If you are running a different distribution, then you should add the text of the files directly into `httpd.conf`.

First is `/etc/apache2/mods-available/auth_memcookie.load`. This file consists of a single line telling Apache to load the Auth MemCookie module.

```
LoadModule mod_auth_memcookie_module
    /usr/lib/apache2/modules/mod_auth_memcookie.so
```

### Note

Note that this should be on a single line. Depending on your distribution, you may have to change the path to `mod_auth_memcookie.so`. To get the path, you can run:

```
apxs2 -q LIBEXECDIR
```

Next, we are going to add the file `/etc/apache2/mods-available/auth_memcookie.conf`, which contains the configuration for Auth MemCookie:

```
<IfModule mod_auth_memcookie.c>
    # '/protected' is the directory which should require access
    # control.
    <Location /protected>
        # This is a list of memcache servers which Auth MemCookie
        # should use. It is a ','-separated list of
```

```
# host:port-pairs.
# Note that this list must list the same servers as the
# 'memcache_servers'-option in config.php in the
# configuration for OpenSSO PHP extension.
Auth_memCookie_Memcached_AddrPort "127.0.0.1:11211"

# This must be set to 'on' to enable Auth MemCookie for
# this directory.
Auth_memCookie_Authoritative on

# These two commands are required to enable access control
# in Apache.
AuthType Cookie
AuthName "My Login"

# This command causes apache to redirect to the given
# URL when we receive a '401 Authorization Required'
# error. We redirect to "/openssophp/spSSOInit.php",
# which initializes a login to the IdP.
ErrorDocument 401 "/openssophp/spSSOInit.php"

# This allows all authenticated users to access the
# directory. To learn more about the 'Require' command,
# please look at:
# http://httpd.apache.org/docs/2.0/mod/core.html#require
Require valid-user
</Location>
</IfModule>
```

If you are running Debian Etch, and have created two files in `/etc/apache2/mods-available/`, then you should enable the module by running:

```
a2enmod auth_memcookie
```

If you added the sections directly into `httpd.conf`, then you don't need to do anything else to enable the module. You still need to restart Apache before the configuration changes have any effect.

You should restart the Apache web server by running:

```
apache2ctl -k restart
```

## Testing the final result

The following code is an example of a test page for Auth MemCookie. It lists all the values which are set by Auth MemCookie. In our example it should be stored as `/var/www/protected/index.php`.

```
<html>
<body>
<table>
<?php
foreach($_SERVER as $key=>$value) {
    if(substr($key, 0, 5) == 'MCAC_') {
        echo('<tr><td>' . $key . '</td><td>' . $value . '</td></tr>');
    }
}
?>
</table>
</body>
```

</html>

Now, if you open `https://sp.example.com/protected/` in your web browser, you should be redirected to the login page on your IdP. After you log in, you should be redirected back to `https://sp.example.com/protected/` and your login information should be listed in the table on that page.

### **Note**

Note that the `authmemcookie-plugin` of the OpenSSO PHP extension urlencodes the values, and your IdP may have base64 encoded the data before returning them to you. It may therefore be necessary to first urldecode the values, and thereafter base64 decode them. Base64 encoding attributes is not normal procedure, but at the time this is written, the Feide federation is doing that.

## **Logging out**

To log a user out from your web site, you will have to redirect the user to the IdP. The precise url may be different on different IdPs. It may be similar to the following URL:

`https://idp.example.com/amserver/IDPSloInit?binding=urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect&RelayState=...`

### **Note**

Note that this URL should be on one line. The `RelayState` parameter tells the IdP which page you want the user to be redirected to after he logs out.