

IN PROGRESS

Customizing Access Control with Darwin Streaming Server

Darwin Streaming Server (DSS) is a common choice for organizations wishing to provide streaming services to their userbase. However, limited support is available for authentication methods other than HTTP Basic, and there is no plugin API to speak of. This document will provide an overview of the modifications necessary to implement new authentication methods, specifically using MySQL to authenticate users against an existing web session database.

Shibboleth Authentication

The original impetus for the DRAM project to modify the DSS source was to provide support for Shibboleth authentication services. The initial design prototype had a significant dependency on Shibboleth, which is why many online references about Shibboleth-DSS integration refer to the DRAM project. However, additional authentication requirements led to development of a more method-neutral login layer, which is the approach we will take here.

Working With the DSS Source

It's important to be prepared for some of the idiosyncrasies of the source DSS distribution. Our modifications are based on version 5.5.5 of the DSS source, and at the time of that release, we found that the `install` script was hugely broken. A number of the installation steps were broken, and all of the paths were hardcoded.

Also, the `Build` script isn't particularly diligent about informing the builder of compilation errors. A quick and dirty solution to double-check your compile is to run `Buildit` a second time, which will only compile items that have failed previously.

The DRAM-DSS distribution has been repaired for the most part, but has only been installed on a limited number of Linux distros, namely Ubuntu (Dapper, Feisty, and Gutsy), Debian, and Red Hat Enterprise Linux 5. It's likely that it will work on most other linux distros, but there may be some changes necessary.

Custom DRAM Access Modules

Basically, there's two different access modules:

APIModules/QTSSAccessModule/ShibMysql.cpp

This is the original access module created by the original DRAM developer. I believe he may have patched the Shibboleth Apache module to save the shib sessions in the MySQL database, but this could have easily been done at the application level, after Shibboleth authentication.

APIModules/QTSSAccessModule/ModuSessionMysql.cpp

This module is probably a better template to use to make your own. I'm not much of a C/C++ programmer, but I cleaned up and modified the original code to work with the session database of the new codebase.

What I've done this time is use Shibboleth as an optional authentication method. Whether our users log in with Shib, IP authentication, or a user/password pair, they get a record in the session table with an assigned `user_id`. Since for us, all logged-in users

should have access to DSS, the access module just verifies that the `user_id` has been defined and that the IP address is the same as the one coming in on the stream request.

Working With Sessions

The latest DRAM-DSS access module (`ModuSessionMysql.cpp`) is built to be auth-method agnostic. It simply validates against a session database populated by your web application. The key to getting this to work properly is the proper abstraction of your web application's authentication code.

Currently DRAM uses a custom Python application layer based on the Twisted framework, but your app can be written in any language, of course.

For example, to do this in PHP should be straightforward. You'll need to have your sessions saved in the database (PHP saves them in a serialized file by default), so you'll have to look into using `session_set_save_handler()` to specify custom callbacks that will write the session info to MySQL. If you're using some kind of framework, like Drupal or Joomla, that should be taken care of already.

If you do want to use the Apache module to provide Shibboleth authentication, you're kind of on your own. I believe that the Shibboleth module adds some stuff to the web server environment (in PHP this is added to the `$_SERVER` array) that you can use to see if the login was successful, and authenticate your sessions that way. This type of approach will allow you to provide alternate authentication mechanisms, which at the very least is useful for local development and testing, or when a Shibboleth IdP server is unavailable.

You should probably modify/emulate the `ModuSessionMysql` access module, as it will do everything you want, and is written more securely. You'll just need to make sure the queries are using the right field names, etc. Also, the name of the parameter I appended to the DSS URL was 'sess', if you want to use something else you'll just need to modify that (in `QTSSAccessModule.cpp`, Line ~ 673).

Here's the schema for the `session` table that the `ModuSessionMysql` access module reads from:

Field	Type	Null	Key	Default	Extra
<code>id</code>	<code>varchar(255)</code>	NO	PRI		
<code>user_id</code>	<code>bigint(20)</code>	YES	MUL	NULL	
<code>created</code>	<code>int(11)</code>	YES		NULL	
<code>accessed</code>	<code>int(11)</code>	YES	MUL	NULL	
<code>timeout</code>	<code>int(11)</code>	YES	MUL	NULL	
<code>data</code>	<code>blob</code>	YES		NULL	
<code>client_ip</code>	<code>varchar(255)</code>	YES		NULL	

The key fields used by the authentication code are:

id

the session cookie value

user_id

the logged-in user id, or 0, or NULL

client_ip

the client's IP address

There's also a table of current DSS sessions. I'm not really sure what this is for, but I think DSS uses it as an authentication cache.

Field	Type	Null	Key	Default	Extra
client_ip	varchar(255)	YES	MUL	NULL	
darwin_session	int(11)	YES	MUL	NULL	

File Modification Overview

This is a list of the files normally in the stock DSS distribution that have been modified in the DRAM-DSS distribution.

`streamingserver.xml`

Added preferences to specify DSS database access, used by access module. (Line ~ 530)

`WebAdmin/src/streamingadminserver.pl`

Numerous fixes, and changed the default paths to keep everything in `/usr/local/DarwinStreamingServer-x.x.x`. Ideally this would be modified to use a prefix variable, so you could change the installation path easily.

`Install`

Many fixes for file locations, changes to default path

`Makefile.POSIX`

Added the new modules to the build list, included the MySQL header files. (Lines ~ 45, 51, 134)

`APIStubLib/QTSS.h`

Created constants for new access modules. (Line ~180)

`APICommonCode/QTAccessFile.cpp`

Implement new 'keywords' for specifying the authentication type in the `qtaccess` file. (Line ~ 400)

`APIModules/QTSSAccessModule/QTSSAccessModule.cpp`

Fetch the preferences assigned in the XML conf file, instantiate the new access modules, and call appropriate functions for authentication.

`Server.tproj/QTSServerPrefs.cpp`

Added conditionals to compare chosen auth scheme to available auth methods.

`Server.tproj/RTSPSession.cpp`

Added conditionals to compare chosen auth scheme to available auth methods.