Trac and mod_wsgi

?mod_wsgi is an Apache module for running WSGI-compatible Python applications directly on top of the Apache webserver. The mod_wsgi adapter is written completely in C and provides very good performances.

Overview

1. The `trac.wsgi` script
    1. A very basic script
    2. A more elaborate script
    3. Recommended `trac.wsgi` script
2. Mapping requests to the script
3. Configuring Authentication
    1. Using Basic Authentication
    2. Using Digest Authentication
    3. Using LDAP Authentication
    4. Using SSPI Authentication
    5. Using Apache authentication with the Account Manager plugin's Login form
    6. Example: Apache/mod_wsgi with Basic Authentication, Trac being at the root of a virtual host
4. Troubleshooting
    1. Use a recent version
    2. Getting Trac to work nicely with SSPI and 'Require Group'
    3. Trac with PostgreSQL
    4. Other resources

The `trac.wsgi` script

Trac can be run on top of mod_wsgi with the help of the following application script, which is just a Python file, though usually saved with a `.wsgi` extension).

A very basic script

In its simplest form, the script could be:

```
import os

os.environ['TRAC_ENV'] = '/usr/local/trac/mysite'
os.environ['PYTHON_EGG_CACHE'] = '/usr/local/trac/mysite/eggs'

import trac.web.main
application = trac.web.main.dispatch_request
```

The `TRAC_ENV` variable should naturally be the directory for your Trac environment (if you have several Trac environments in a directory, you can also use `TRAC_ENV_PARENT_DIR` instead), while the `PYTHON_EGG_CACHE` should be a directory where Python can temporarily extract Python eggs.

A more elaborate script

If you're using multiple `.wsgi` files (for example one per Trac environment) you must *not* use `os.environ['TRAC_ENV']` to set the path to the Trac environment. Using this method may lead to Trac delivering the content of another Trac environment, as the variable may be filled with the path of a previously viewed Trac environment.

To solve this problem, use the following `.wsgi` file instead:

```
import os

os.environ['PYTHON_EGG_CACHE'] = '/usr/local/trac/mysite/eggs'

import trac.web.main
def application(environ, start_response):
  environ['trac.env_path'] = '/usr/local/trac/mysite'
  return trac.web.main.dispatch_request(environ, start_response)
```

For clarity, you should give this file a `.wsgi` extension. You should probably put the file in its own directory, since you will expose it to Apache.

If you have installed Trac and eggs in a path different from the standard one you should add that path by adding the following code at the top of the wsgi script:

```
import site
site.addsitedir('/usr/local/trac/lib/python2.4/site-packages')
```

Change it according to the path you installed the Trac libs at.

Recommended `trac.wsgi` script

A somewhat robust and generic version of this file can be created using the `trac-admin <env> deploy <dir>` command which automatically substitutes the required paths (see TracInstall#cgi-bin).

Mapping requests to the script

After you've done preparing your .wsgi script, add the following to your Apache configuration file (`httpd.conf` for example).

```
WSGIScriptAlias /trac /usr/local/trac/mysite/apache/mysite.wsgi

<Directory /usr/local/trac/mysite/apache>
    WSGIApplicationGroup %{GLOBAL}
    Order deny,allow
    Allow from all
</Directory>
```

Here, the script is in a subdirectory of the Trac environment.

If you followed the directions ?Generating the Trac cgi-bin directory, your Apache configuration file should look like following:

```
WSGIScriptAlias /trac /usr/share/trac/cgi-bin/trac.wsgi

<Directory /usr/share/trac/cgi-bin>
    WSGIApplicationGroup %{GLOBAL}
    Order deny,allow
    Allow from all
</Directory>
```

In order to let Apache run the script, access to the directory in which the script resides is opened up to all of Apache. Additionally, the `WSGIApplicationGroup` directive ensures that Trac is always run in the first Python interpreter created by mod_wsgi; this is necessary because the Subversion Python bindings, which are used by Trac, don't always work in other sub-interpreters and may cause requests to hang or cause Apache to crash as a

result. After adding this configuration, restart Apache, and then it should work.

To test the setup of Apache, mod_wsgi and Python itself (ie. without involving Trac and dependencies), this simple wsgi application can be used to make sure that requests gets served (use as only content in your `.wsgi` script):

```
def application(environ, start_response):
        start_response('200 OK',[('Content-type','text/html')])
        return ['<html><body>Hello World!</body></html>']
```

For more information about using the mod_wsgi specific directives, see the ?mod_wsgi's wiki and more specifically the ?IntegrationWithTrac page.

Configuring Authentication

We describe in the the following sections different methods for setting up authentication.

See also ?Authentication, Authorization and Access Control in the Apache guide.

Using Basic Authentication

The simplest way to enable authentication with Apache is to create a password file. Use the `htpasswd` program to create the password file:

```
$ htpasswd -c /somewhere/trac.htpasswd admin
New password: <type password>
Re-type new password: <type password again>
Adding password for user admin
```

After the first user, you dont need the "-c" option anymore:

```
$ htpasswd /somewhere/trac.htpasswd john
New password: <type password>
Re-type new password: <type password again>
Adding password for user john
```

> *See the man page for `htpasswd` for full documentation.*

After you've created the users, you can set their permissions using TracPermissions.

Now, you'll need to enable authentication against the password file in the Apache configuration:

```
<Location "/trac/login">
  AuthType Basic
  AuthName "Trac"
  AuthUserFile /somewhere/trac.htpasswd
  Require valid-user
</Location>
```

If you're hosting multiple projects you can use the same password file for all of them:

```
<LocationMatch "/trac/[^/]+/login">
  AuthType Basic
  AuthName "Trac"
  AuthUserFile /somewhere/trac.htpasswd
  Require valid-user
</LocationMatch>
```

Note that neither a file nor a directory named 'login' needs to exist.
See also the ?mod_auth_basic documentation.

Using Digest Authentication

For better security, it is recommended that you either enable SSL or at least use the ?digest? authentication scheme instead of ?Basic?.

You'll have to create your `.htpasswd` file with the `htdigest` command instead of `htpasswd`, as follows:

```
# htdigest -c /somewhere/trac.htpasswd trac admin
```

The "trac" parameter above is the "realm", and will have to be reused in the Apache configuration in the AuthName directive:

```
<Location "/trac/login">

    AuthType Digest
    AuthName "trac"
    AuthDigestDomain /trac
    AuthUserFile /somewhere/trac.htpasswd
    Require valid-user
</Location>
```

For multiple environments, you can use the same `LocationMatch` as described with the previous method.

Don't forget to activate the mod_auth_digest. For example, on a Debian 4.0r1 (etch) system:

```
    LoadModule auth_digest_module /usr/lib/apache2/modules/mod_auth_digest.so
```

See also the ?mod_auth_digest documentation.

Using LDAP Authentication

Configuration for ?mod_ldap authentication in Apache is a bit tricky (httpd 2.2.x and OpenLDAP: slapd 2.3.19)

1. You need to load the following modules in Apache httpd.conf

```
LoadModule ldap_module modules/mod_ldap.so
LoadModule authnz_ldap_module modules/mod_authnz_ldap.so
```

2. Your httpd.conf also needs to look something like:

```
<Location /trac/>
  # (if you're using it, mod_python specific settings go here)
  Order deny,allow
  Deny from all
  Allow from 192.168.11.0/24
  AuthType Basic
  AuthName "Trac"
  AuthBasicProvider "ldap"
  AuthLDAPURL "ldap://127.0.0.1/dc=example,dc=co,dc=ke?uid?sub?(objectClass=inetOrgPerson)"
  authzldapauthoritative Off
  Require valid-user
</Location>
```

3. You can use the LDAP interface as a way to authenticate to a Microsoft Active Directory:

Use the following as your LDAP URL:

```
AuthLDAPURL "ldap://directory.example.com:3268/DC=example,DC=com?sAMAccountName?sub?(objectClass=us
```

You will also need to provide an account for Apache to use when checking credentials. As this password will be listed in plaintext in the config, you should be sure to use an account specifically for this task:

```
AuthLDAPBindDN ldap-auth-user@example.com
AuthLDAPBindPassword "password"
```

The whole section looks like:

```
<Location /trac/>
  # (if you're using it, mod_python specific settings go here)
  Order deny,allow
  Deny from all
  Allow from 192.168.11.0/24
  AuthType Basic
  AuthName "Trac"
  AuthBasicProvider "ldap"
  AuthLDAPURL "ldap://adserver.company.com:3268/DC=company,DC=com?sAMAccountName?sub?(objectClass=user)
  AuthLDAPBindDN       ldap-auth-user@company.com
  AuthLDAPBindPassword "the_password"
  authzldapauthoritative Off
  # require valid-user
  require ldap-group CN=Trac Users,CN=Users,DC=company,DC=com
</Location>
```

Note 1: This is the case where the LDAP search will get around the multiple OUs, conecting to Global Catalog Server portion of AD (Notice the port is 3268, not the normal LDAP 389). The GCS is basically a "flattened" tree which allows searching for a user without knowing to which OU they belong.

Note 2: You can also require the user be a member of a certain LDAP group, instead of just having a valid login:

```
Require ldap-group CN=Trac Users,CN=Users,DC=example,DC=com
```

See also:

- ?mod_authnz_ldap, documentation for mod_authnz_ldap
- ?mod_ldap, documentation for mod_ldap, which provides connection pooling and a shared cache.
- ?TracHacks:LdapPlugin for storing TracPermissions in LDAP.

Using SSPI Authentication

If you are using Apache on Windows, you can use mod_auth_sspi to provide single-sign-on. Download the module from the SourceForge ?mod-auth-sspi project and then add the following to your VirtualHost:

```
<Location /trac/login>
    AuthType SSPI
    AuthName "Trac Login"
    SSPIAuth On
    SSPIAuthoritative On
    SSPIDomain MyLocalDomain
    SSPIOfferBasic On
    SSPIOmitDomain Off
    SSPIBasicPreferred On
    Require valid-user
</Location>
```

Using the above, usernames in Trac will be of the form `DOMAIN\username`, so you may have to re-add permissions and such. If you do not want the domain to be part of the username, set `SSPIOmitDomain On` instead.

Some common problems with SSPI authentication: ?#1055, ?#1168 and ?#3338.

See also ?TracOnWindows/Advanced.

Using Apache authentication with the Account Manager plugin's Login form

To begin with, see the basic instructions for using the Account Manager plugin's ?Login module and its ?HttpAuthStore authentication module.

Note: If is difficult to get HttpAuthStore to work with WSGI when using any Account Manager version prior to acct_mgr-0.4. Upgrading is recommended.

Here is an example (from the HttpAuthStore link) using acct_mgr-0.4 for hosting a single project:

```
[components]
; be sure to enable the component
acct_mgr.http.HttpAuthStore = enabled

[account-manager]
; configure the plugin to use a page that is secured with http authentication
authentication_url = /authFile
password_store = HttpAuthStore
```

This will generally be matched with an Apache config like:

```
<Location /authFile>
   ?HTTP authentication configuration?
   Require valid-user
</Location>
```

Note that **authFile** need not exist. See the HttpAuthStore link above for examples where multiple Trac projects are hosted on a server.

Example: Apache/mod_wsgi with Basic Authentication, Trac being at the root of a virtual host

Per the mod_wsgi documentation linked to above, here is an example Apache configuration that a) serves the Trac instance from a virtualhost subdomain and b) uses Apache basic authentication for Trac authentication.

If you want your Trac to be served from e.g. http://trac.my-proj.my-site.org, then from the folder e.g. `/home/trac-for-my-proj`, if you used the command `trac-admin the-env initenv` to create a folder `the-env`, and you used `trac-admin the-env deploy the-deploy` to create a folder `the-deploy`, then first:

Create the htpasswd file:

```
cd /home/trac-for-my-proj/the-env
htpasswd -c htpasswd firstuser
### and add more users to it as needed:
htpasswd htpasswd seconduser
```

(keep the file above your document root for security reasons)

Create this file e.g. (ubuntu) `/etc/apache2/sites-enabled/trac.my-proj.my-site.org.conf` with the following contents:

```
<Directory /home/trac-for-my-proj/the-deploy/cgi-bin/trac.wsgi>
  WSGIApplicationGroup %{GLOBAL}
  Order deny,allow
  Allow from all
</Directory>

<VirtualHost *:80>
  ServerName trac.my-proj.my-site.org
  DocumentRoot /home/trac-for-my-proj/the-env/htdocs/
  WSGIScriptAlias / /home/trac-for-my-proj/the-deploy/cgi-bin/trac.wsgi
  <Location '/'>
    AuthType Basic
    AuthName "Trac"
    AuthUserFile /home/trac-for-my-proj/the-env/htpasswd
    Require valid-user
  </Location>
</VirtualHost>
```

Note: for subdomains to work you would probably also need to alter `/etc/hosts` and add A-Records to your host's DNS.

Troubleshooting

Use a recent version

Please use either version 1.6, 2.4 or later of `mod_wsgi`. Versions prior to 2.4 in the 2.X branch have problems with some Apache configurations that use WSGI file wrapper extension. This extension is used in Trac to serve up attachments and static media files such as style sheets. If you are affected by this problem attachments will appear to be empty and formatting of HTML pages will appear not to work due to style sheet files not loading properly. Another frequent symptom is that binary attachment downloads are truncated. See mod_wsgi tickets ?#100 and ?#132.

Note: using mod_wsgi 2.5 and Python 2.6.1 gave an Internal Server Error on my system (Apache 2.2.11 and Trac 0.11.2.1). Upgrading to Python 2.6.2 (as suggested ?here) solved this for me
-- Graham Shanks

If you plan to use `mod_wsgi` in embedded mode on Windows or with the MPM worker on Linux, then you'll even need version 0.3.4 or greater (see ?#10675 for details).

Getting Trac to work nicely with SSPI and 'Require Group'

If like me you've set Trac up on Apache, Win32 and configured SSPI, but added a 'Require group' option to your apache configuration, then the SSPIOmitDomain option is probably not working. If its not working your usernames in trac are probably looking like 'DOMAIN\user' rather than 'user'.

This WSGI script 'fixes' things, hope it helps:

```
import os
import trac.web.main

os.environ['TRAC_ENV'] = '/usr/local/trac/mysite'
os.environ['PYTHON_EGG_CACHE'] = '/usr/local/trac/mysite/eggs'
```

```
def application(environ, start_response):
    if "\\" in environ['REMOTE_USER']:
        environ['REMOTE_USER'] = environ['REMOTE_USER'].split("\\", 1)[1]
    return trac.web.main.dispatch_request(environ, start_response)
```

Trac with PostgreSQL

When using the mod_wsgi adapter with multiple Trac instances and PostgreSQL (or MySQL?) as a database back-end, the server *may* create a lot of open database connections and thus PostgreSQL processes.

A somewhat brutal workaround is to disabled connection pooling in Trac. This is done by setting `poolable = False` in `trac.db.postgres_backend` on the `PostgreSQLConnection` class.

But it's not necessary to edit the source of Trac, the following lines in `trac.wsgi` will also work:

```
import trac.db.postgres_backend
trac.db.postgres_backend.PostgreSQLConnection.poolable = False
```

or

```
import trac.db.mysql_backend
trac.db.mysql_backend.MySQLConnection.poolable = False
```

Now Trac drops the connection after serving a page and the connection count on the database will be kept minimal.

This is not a recommended approach though. See also the notes at the bottom of the ?mod_wsgi's IntegrationWithTrac wiki page.

Other resources

For more troubleshooting tips, see also the mod_python troubleshooting section, as most Apache-related issues are quite similar, plus discussion of potential ?application issues when using mod_wsgi. The wsgi page also has a ?Integration With Trac document.

See also: TracGuide, TracInstall, FastCGI, ModPython, ?TracNginxRecipe