

A shim based approach to authentication using CAS.

(It works for authorization too.)

Erik Klavon

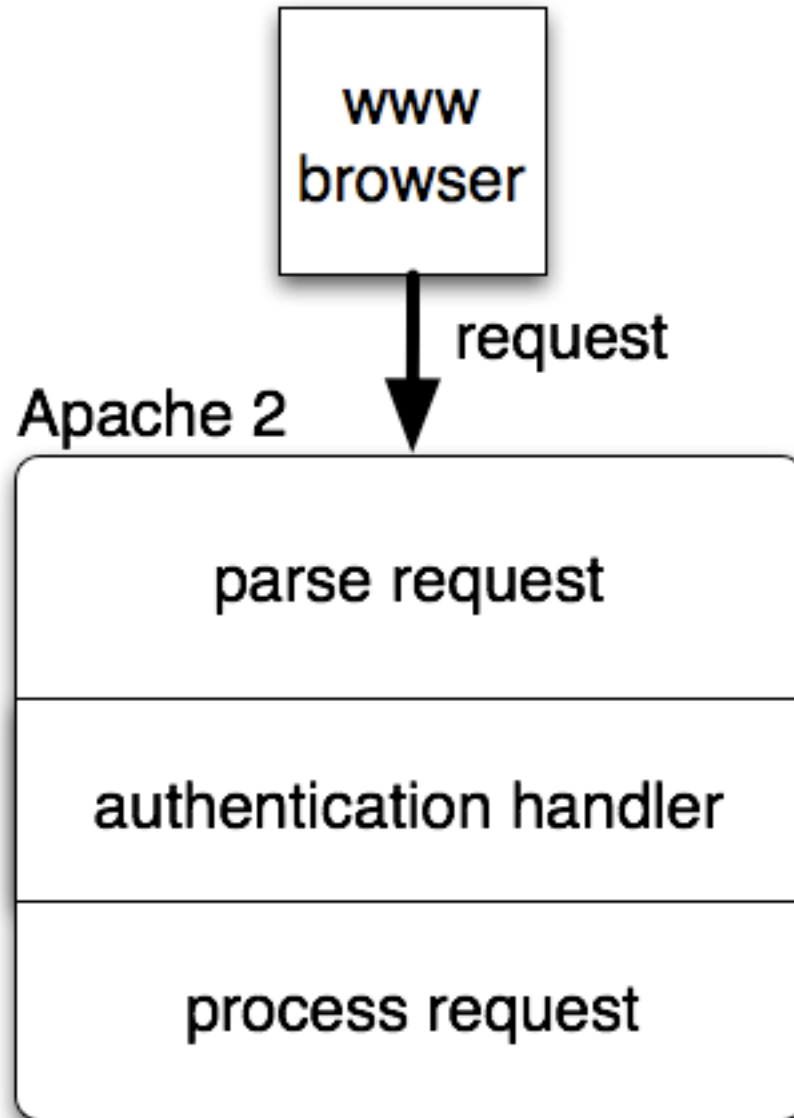
erik@ack.berkeley.edu

2008-10-08

My authentication needs

- Home grown web applications served by Apache 2 and written in perl (some use AWS, others basic auth).
- Third party web applications with different authentication mechanisms.
 - MoinMoin (Python)
 - Nagios (C)
 - Cacti (php)
 - Request Tracker (perl)
- Static web pages protected by basic authentication.
- The ideal solution should work for all cases.

Apache 2 authentication handler



Use an authentication handler to implement CAS authentication.

- Requirements:
 - Track user sessions. (Use a cookie and keep state in the server).
 - When an unauthenticated user makes a request, redirect them to the CAS login page. (Apache 2 handlers support redirection.)
 - Validate CAS authentication when present and user does not have a session. (This covers a return redirect after CAS login as well as single sign on.)
 - Identify the authenticated user to web applications protected by the handler. (Use REMOTE_USER environment variable.)
 - Selectively enforce authentication with the granularity of a URL. (Use Apache 2 configuration directives to control where authentication is required and where it is not.)

Use existing solutions

- Apache2::AuthCAS (mod_perl) and mod_auth_cas (C) both meet these requirements.
- Both can store the user identity returned from CAS (for UCB the CalNet directory UID) in the REMOTE_USER environment variable.
- I started out using Apache2::AuthCAS and am evaluating mod_auth_cas.
- Apache2::AuthCAS known to work with C, php, python and perl web apps as well as static content. mod_auth_cas should be similar; it works fine with perl.

Testing CAS Integration

- Cases
 - User authenticates for the first time
 - Single sign on
 - CAS session times out
 - Shim/App session times out
 - Both CAS and Shim/App sessions time out
- For each case, how are POSTs and GETs handled?
- You may want to avoid exposing user submitted data to the CAS servers when using GET.

Apache2::AuthCAS vs. mod_auth_cas

	AuthCAS	mod_auth_cas
Session State	Client cookie and SQL database	Client cookie and local filesystem
Validate cert of CAS server during auth validation	No	Yes
Handles POST w/o data loss across authen+	No	No
Supports proxy functions	Yes (untested)	No

Apache2::AuthCAS example

```
PerlLoadModule Apache2::Request
PerlLoadModule Apache2::AuthCAS::Configuration
PerlLoadModule Apache2::AuthCAS

<Location "/usr/www/sec-cgi-bin/hello_world/">
  AuthType Apache2::AuthCAS

  AuthName "CAS"
  PerlAuthenHandler Apache2::AuthCAS->authenticate
  require valid-user

  CASDbDriver          "Pg"
  CASDbDataSource      "dbname=<db>;host=<host>;port=<port>"
  CASDbUser            "<username>"
  CASDbPass            "<passwd>"

  CASHost              "auth.berkeley.edu"
  CASServiceValidateUri "/cas/serviceValidate"

  CASPretendBasicAuth  1
</Location>
```


mod_auth_cas example

```
CASVersion 2
CASLoginURL https://auth.Berkeley.EDU/cas/login
CASValidateURL https://auth.Berkeley.EDU/cas/serviceValidate
CASCookieDomain net.berkeley.edu
CASCertificatePath /usr/local/ist/etc/ssl/certs/auth.pem

<Location "/usr/www/sec-cgi-bin/hello_world/">
    AuthType CAS
    require valid-user
</Location>
```

Obtaining user identity example

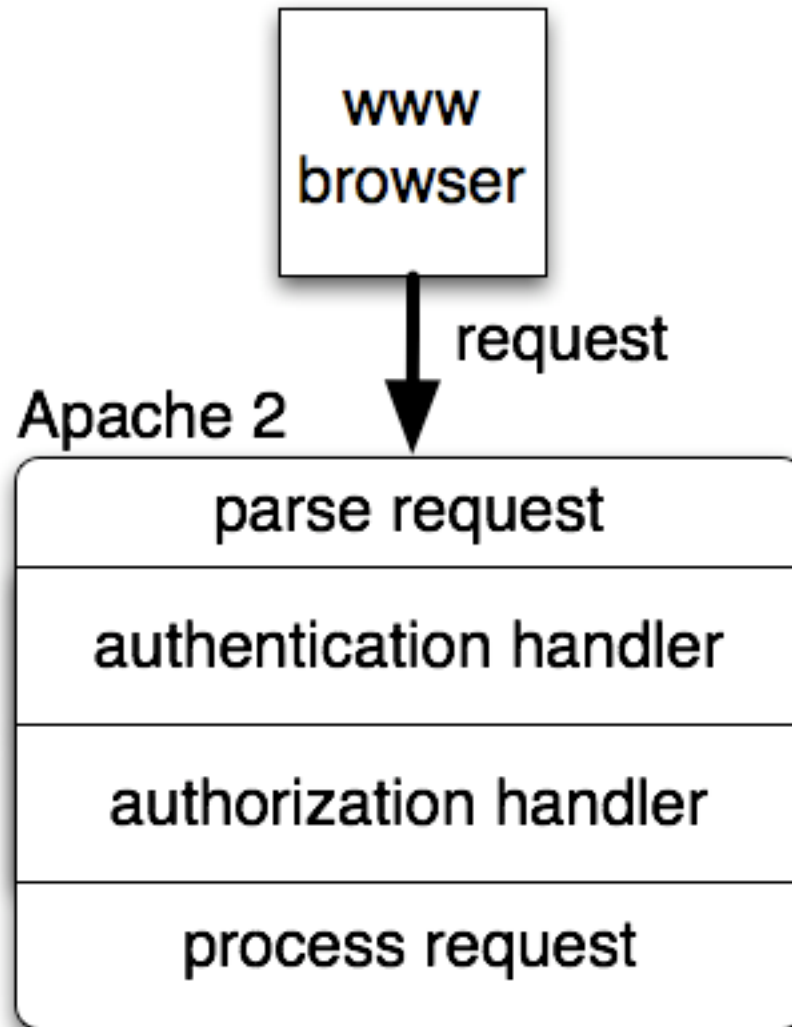
```
#!/usr/bin/perl

use CGI;

my $cgi = new CGI();
my $calnetuid = $cgi->remote_user();

if (!defined($calnetuid) || ($calnetuid eq '')) {
    # need to handle auth error case here; display
    # error page to user.
}
```

Use an authorization handler to implement authorization



Apache 2 authorization handlers

- Numerous versions exist.
- I wrote my own in mod_perl to meet requirements of our environment.
 - Role based authorization against Unix account (group) data.
 - Per user authorization by CalNet UID.
 - Rewrite REMOTE_USER variable from CalNet UID to some other identifier on a per application basis when needed.
 - I may add the ability to perform authorization by applying criteria against CalNet directory info.
 - mod_authz_ldap may work for CalNet directory authorization.
- This works really well for static content and many web applications.
- Depending on the situation, you may want to perform some authorization in your application rather than in the web server.

Authorization example

```
PerlLoadModule IST::Apache2::AuthzLDAP::Configuration  
PerlLoadModule IST::Apache2::AuthzLDAP
```

```
<Location "/usr/www/sec-cgi-bin/hello_world/">  
    PerlAuthzHandler IST::Apache2::AuthzLDAP->handler  
    AuthzLDAPLDAPServer "<server1>,<server2>"  
    AuthzLDAPLDAPBind "<bind dn>"  
    AuthzLDAPLDAPPasswd "<bindpasswd>"  
    AuthzLDAPLogLevel "4"  
    AuthzLDAPGroup "staff,wheel"  
    AuthzLDAPCalNetUID "106466"  
    AuthzLDAPRemoteUserType "FirstLast"  
</Location>
```

Conclusion

- Authentication and authorization handlers are shims that selectively modify Apache's behavior.
- You can download an authentication handler to implement CAS authentication for static content and most web applications. No coding required!
- You can download or write your own authorization handler to authorize access to static content and web applications.
- These are configure/write once solutions. All the work is taken care of in the web server; your applications do not need to be concerned with authentication or authorization (where applicable).
- Centralizing authentication and authorization functions in the web server makes it easier to develop and deploy work arounds when CAS or authorization data are unavailable.
- Something similar may be possible for IIS.