Module Apache

Table des matières

1 Intro

- Writing Apache Modules With Perl And C
- http://perl.apache.org/docs/2.0
- http://httpd.apache.org/apreq/docs/libapreq2/group___apreq___xs___cookie.html
- https://debian-administration.org/article/635/A_brief_introduction_to_mod_perl_-_Part_1
- htmladdnormallinkhttps ://perl.apache.org/docs/2.0/api/Apache2/Log.htmlhttps ://perl.apache.org/docs/2.0/api

```
# apt-get install libapache2-mod-perl2 libapache2-mod-perl2-doc \
                  libapache2-request-perl libapreq2-doc \
                  libapache2-authcookie-perl

$ apt-cache policy apache2
  Installé : 2.4.10-10+deb8u7

$ apt-cache policy libapache2-mod-perl2
  Installé : 2.0.9~1624218-2+deb8u1

$ man Apache2::Cookie
```

2 Hello World

- */etc/apache2/sites-enabled/000-default.conf*

```
<Perl>
    use lib '/home/nroche/modperl';
</Perl>

<Location /perl-status>
    SetHandler perl-script
    PerlHandler +Apache2::Status
 </Location>

PerlModule Apache2::Hello
<Location /hello>
  SetHandler perl-script
  PerlResponseHandler Apache2::Hello
</Location>
```

- */home/nroche/modperl/Apache2/Hello.pm*

```
package Apache2::Hello;

use Apache2::RequestRec;
use Apache2::RequestIO;
use Apache2::Const -compile => qw(OK);

sub handler {
    my $r = shift;

    $r->content_type('text/html');
    $r->print('Hello World!');

    return Apache2::Const::OK;
}

1;
```

3 Ecrire et lire un cookie

- */usr/bin/cgi-bin/hello.pm*

```
#!/usr/bin/perl -wT

use strict;
use warnings;
use utf8;
use CGI;
use CGI::Cookie;

my $query = CGI->new; # create new CGI object

# Create new cookies and send them
my $cookie = $query->cookie(
    -name=>'mediatex',
    -value=>'coll1:index,cache;coll2:index',
    -domain  =>  '.localhost',
    -expires=>'+5m',
    -path=>'/');

print $query->header('text/html', -cookie=>$cookie);
```

```
print $query->start_html('My cookie-set.cgi program');
print $query->h3('The cookie has been set');
print $query->end_html;

exit;
```

- */home/nroche/modperl/Apache2/Hello.pm*

```
package Apache2::Hello;

use Apache2::RequestRec;
use Apache2::RequestIO;
use Apache2::Cookie;
use Apache2::Const -compile => qw(OK);

sub handler {
    my $r = shift;
    my $req = $r->pool();

    # fetch existing cookies
    my %cookie = Apache2::Cookie->fetch($r);


    $r->content_type('text/html');
    $r->print('Hello World!<br>');

    foreach my $k (keys(%cookie)) {
print "<hr>Clef: '$k'<br>\n";
print "Valeur: '$cookie{$k}'<hr>\n";
    }

    return Apache2::Const::OK;
}

1;
```

- Dé-saussissoner une chaîne `coll1:index,cache;coll2:index`

```
#!/usr/bin/perl -wT
use strict;
use warnings;

my $string = "coll1:index,cache;coll2:index";

my @stanzas = split(/;/, $string);
foreach my $stanza (@stanzas) {

    my @parts = split(/:/, $stanza);
    my $coll = $parts[0];
    my $part2 = $parts[1];
    print $coll."\n";

    @perms = split(/,/, $part2);
    foreach my $perm (@perms) {
print "- ".$perm."\n";
    }
}
```

4 Matcher avec une indication en conf d'apache

- */var/www/html/hello/index.html*

```
<h1>
Hello
</h1>
```

- */etc/apache2/sites-enabled/000-default.conf*

```
<Perl>
    use lib '/home/nroche/modperl';
</Perl>

PerlModule Apache2::Hello
<Directory /var/www/html/hello>
  SetHandler perl-script
  PerlResponseHandler Apache2::Hello
  PerlSetVar COLLECTION coll2
  PerlSetVar GROUP index
</Directory>
```

- */home/nroche/modperl/Apache2/Hello.pm*

```
package Apache2::Hello;

use Apache2::RequestRec;
use Apache2::RequestIO;
use Apache2::Cookie;
use Apache2::ServerRec;
use Apache2::Const -compile => qw(DECLINED OK SERVER_ERROR);

sub handler {
    my $r = shift;
    my $req = $r->pool();

    # get configuration values
    my $confColl=$r->dir_config('COLLECTION');
    my $confGroup=$r->dir_config('GROUP');

    if ($confColl eq '') {
$r->log_error("COLLECTION variable unset");
return Apache2::Const::SERVER_ERROR;
    }

    if ($confGroup eq '') {
$r->log_error("GROUP variable unset");
return Apache2::Const::SERVER_ERROR;
    }

    # fetch existing cookies
    my %cookies = Apache2::Cookie->fetch($r);
    my $cookie = $cookies{"mediatex"};
    if ($cookie eq '') {
$r->content_type('text/html');
$r->print('Hello '.$confColl.'!<br>');
$r->log_error("pas de cookie pour mediatex");
return Apache2::Const::OK;
    }
```

```
        # split cookie string
        # "mediatex=coll1:index,cache;coll2:index
        my @parts = split(/=/, $cookie);
        my $string = $parts[1];

        # coll1:index,cache;coll2:index
        my @perms = split(/%3B/, $string);
        foreach my $perm (@perms) {

    # coll1:index,cache
    @parts = split(/%3A/, $perm);
    my $coll = $parts[0];
    my $string = $parts[1];

    if ($confColl eq $coll) {
        # index,cache
        @groups = split(/%2C/, $string);
        foreach my $group (@groups) {
    if ($confGroup eq $group) {
        return Apache2::Const::DECLINED;
    }
        }
    }
        }

        $r->log_error("pas les bon droits pour ".
      $confColl.":".$confGroup.".<br>");
        return Apache2::Const::OK;
    }

    1;
```

5 Crypter/Décrypter

- http://www.perl.com/pub/2001/09/26/crypto1.html
- http://stuff-things.net/2007/05/02/encrypting-sensitive-data-with-perl/

```
# apt-get install libcrypt-openssl-rsa-perl libconvert-pem-perl

$ openssl genrsa -des3 -out private.pem 2048
$ openssl rsa -in private.pem -out public.pem -outform PEM -pubout

// ne marche pas ici
$ openssl req -nodes -new -x509 -keyout sso.key -out sso.cert
```

encrypt.pl

```
#!/usr/bin/perl -wT

use strict;
use warnings;

use Crypt::OpenSSL::RSA;
use MIME::Base64;
use strict;
```

```
my $public_key = 'public.pem';
my $string = 'Hello World!';

print encryptPublic($public_key,$string);

exit;

sub encryptPublic {
  my ($public_key,$string) = @_;

  my $key_string;
  open(PUB,$public_key) || die "$public_key: $!";
  read(PUB,$key_string,-s PUB); # Suck in the whole file
  close(PUB);

  my $public =
          Crypt::OpenSSL::RSA->new_public_key($key_string);
  encode_base64($public->encrypt($string));
}
```

decrypt.pl

```
#!/usr/bin/perl -wT

use strict;
use warnings;

use Convert::PEM;
use Crypt::OpenSSL::RSA;
use MIME::Base64;

my $encrypted_string =q(
Df34vuBnYGx4m6DJUG2gmsufAposm9KW5ZC6IQzDp5bJ1O5/gCNL5aZBT8lO3noijb6+kK9ogrfX
dkXR4EFlr2ARjawVS9fqjAj2cgirqoaw9FUl5xQISakazOf6wuTkTsBt1KxjX2CF/NRBjdduBXHR
a3c72aAu8Wxqz9FCxxaAVqtlNuEaWw/PnDpC7KE4QWqvf7jlGORg1VQItpL5FmelUjCyfW7WcSku
AsrZ97U3e9xyv0pQwAAJdaqPAjNPTYzD9fzp5v1vOOeQb/B9qWbYG49urC5WSBi1GogmnJL1841v
VWAaU/iaycvviOCNq+7flKeg27mTqKE7uflubw==
);

my $private_key = 'private.pem';
my $password = '1234';

print decryptPrivate($private_key,$password,$encrypted_string),  "\n";

exit;

sub decryptPrivate {
  my ($private_key,$password,$string) = @_;
  my $key_string = readPrivateKey($private_key,$password);

  return(undef) unless ($key_string); # Decrypt failed.
  my $private = Crypt::OpenSSL::RSA->new_private_key($key_string) ||
  die "$!";

  $private->decrypt(decode_base64($string));
}

sub readPrivateKey {
  my ($file,$password) = @_;
```

```
  my $key_string;
  $key_string = decryptPEM($file,$password);
}

sub decryptPEM {
  my ($file,$password) = @_;

  my $pem = Convert::PEM->new(
      Name => 'RSA PRIVATE KEY',
      ASN  => qq(
                  RSAPrivateKey SEQUENCE {
                      version INTEGER,
                      n INTEGER,
                      e INTEGER,
                      d INTEGER,
                      p INTEGER,
                      q INTEGER,
                      dp INTEGER,
                      dq INTEGER,
                      iqmp INTEGER
                  }
           ));

  my $pkey =
      $pem->read(Filename => $file, Password => $password);

  return(undef) unless ($pkey); # Decrypt failed.
  $pem->encode(Content => $pkey);
}
```

6 Formulaire d'authentification

- http://www.perlmeme.org/tutorials/cgi_form.html
- http://perl.mines-albi.fr/ModulesFr/CGI.html

/usr/bin/cgi-bin/authentication.pm

```
#!/usr/bin/perl

use strict;
use warnings;
use CGI;
use CGI::Carp qw(fatalsToBrowser); # Remove this in production

my $q = new CGI;
my $target_url = "";
my $bad_login = 0;
my $good_login = 0;

my $login = $q->param('login');
my $passwd = $q->param('passwd');
my $url = $q->param('url');

check_results($q);

output_top($q);
output_form($q);
```

```
output_end($q);

exit 0;

#---------------------------------------------------------------

# Check the results of the form
sub check_results {
    my ($q) = @_;

    if ($login eq "" && $passwd eq "") {
return;
    }

    if ($login eq "toto" && $passwd eq "tata") {
$good_login = 1;
    }
    else {
$bad_login = 1;
    }

    if ($good_login eq 1 && $url ne "") {
print $q->redirect(-uri=>$url);
    }
}

# Outputs the start html tag, stylesheet and heading
sub output_top {
    my ($q) = @_;

    print $q->header();

    print $q->start_html(
-title => 'Autentication:',
-bgcolor => 'white',
-style => {
    -code => '
                    /* Stylesheet code */
                    body {
                        font-family: verdana, sans-serif;
                    }
                    h2 {
                        color: darkblue;
                        border-bottom: 1pt solid;
                        width: 100%;
                    }
                    h4 {
                        color: red;
                    }
                    h5 {
                        color: blue;
                    }
                    div {
                        text-align: right;
                        color: steelblue;
                        border-top: darkblue 1pt solid;
                        margin-top: 4pt;
                    }
```

```
                    th {
                        text-align: right;
                        padding: 2pt;
                        vertical-align: top;
                    }
                    td {
                        padding: 2pt;
                        vertical-align: top;
                    }
                    /* End Stylesheet code */
                ’,
},
        );

    print $q->h2("Authentication");
    if ($bad_login eq 1) {
print $q->h4(’ Bad credentials, please try again’);
    }
    if ($good_login eq 1) {
print $q->h5(’ Good credentials (no redirection priovided)’);
    }
}

# Outputs a web form
sub output_form {
    my ($q) = @_;
    print $q->start_form(
-name => ’main’,
-method => ’POST’,
        );

    print $q->start_table;
    print $q->Tr(
$q->td(’Login:’),
$q->td(
            $q->textfield(-name => "login", -size => 50)
)
        );
    print $q->Tr(
$q->td(’Password:’),
$q->td(
            $q->password_field(-name => "passwd", -size => 50)
)
);
    print $q->hidden(-name => "url", -default => $url);
    print $q->Tr(
$q->td($q->submit(-value => ’Submit’)),
$q->td(’ ’)
        );
    print $q->end_table;
    print $q->end_form;
}

# Outputs a footer line and end html tags
sub output_end {
    my ($q) = @_;
    print $q->div("Single sign one");
    print $q->end_html;
```

```
}
```

7 Informations stockées via Phpbb

7.1 Vérification du mot de passe

PhpBB utilise le projet `PHPass` pour hasher les mots de passe (en utilisant un 'P' à la place du 'H' comme identifiant de fonction de hashage). Il existe un podule Perl adéquat.

- PHPass
- PHPass.pm

```
# apt-get install libauthen-passphrase-perl
```

```
#!/usr/bin/perl

use strict;
use warnings;
use utf8;

# apt-get install libauthen-passphrase-perl
use Authen::Passphrase::PHPass;

my $passwd1 = "xxxxxxxx";
my $passwd2 = '$H$9Zo7eE/nWtXnHh5iobQ9h5EWDMYRhg/';

substr($passwd2, 1, 1) = "P";
my $ppr = Authen::Passphrase::PHPass->from_crypt($passwd2);

if($ppr->match($passwd1)) {
    print "match\n";
}
else {
    print "do not match\n";
}

exit 0
```

7.2 Récupération des permissions

On va récupérer nos informations dans la description des groupes de l'utilisateurs.

```
#!/usr/bin/perl

use strict;
use warnings;
use utf8;
use CGI;
use DBI;
use Authen::Passphrase::PHPass;

my $q = new CGI;

# database
my $dbHost="localhost";
my $dbPort="5432";
my $dbName = "phpbb3";
```

```
my $dbUser = "phpbb3";
my $dbPasswd = "...";

# incoming parameters
my $login = "...";
my $passwd = "...";
my $url = $q->param('url');
my $permissions = "";

output_top($q);
check_passwd($q);
print "<br>permissions = $permissions<br>";
output_end($q);

exit 0;

# Check the results of the form
sub check_passwd {
    my ($q) = @_;
    my $query1 = "select user_password from phpbb_users where username='".
$login."'";
    my $query2 = "select group_desc".
" from phpbb_users, phpbb_user_group, phpbb_groups".
" where username = '".$login."'".
" and phpbb_users.user_id = phpbb_user_group.user_id".
" and phpbb_groups.group_id = phpbb_user_group.group_id".
" and group_name like 'mdtx-%'";
    my $dbh;
    my $cursor;
    my @row;
    my $ppr;

    # connect DB
    if (!($dbh = DBI->connect(
           "dbi:Pg:dbname=$dbName;host=$dbHost;port=$dbPort",
           $dbUser, $dbPasswd
       ))) {
     printf "Database error: ".$DBI::errstr;
     return;
    }

    # check passwd
    $cursor = $dbh->prepare($query1);
    $cursor->execute;

    @row = $cursor->fetchrow;
    substr($row[0], 1, 1) = "P";
    $ppr = Authen::Passphrase::PHPass->from_crypt($row[0]);
    if($ppr->match($passwd)) {
print "match\n";
    }
    else {
print "do not match\n";
    }
    $cursor->finish;

    # get groups
    $cursor = $dbh->prepare($query2);
```

```
    $cursor->execute;

    $permissions="";
    while (@row = $cursor->fetchrow) {
$permissions=$permissions.$row[0].';';
    }

    $cursor->finish;
    $dbh->disconnect;
}
...
```

8 Redirections

```
    if ($cookie eq '') {
$r->log_error("pas de cookie pour mediatex");
$r->content_type('text/html');
$r->err_headers_out->add('Location' => 'http://localhost/cgi-bin/hello.pm');
return Apache2::Const::REDIRECT;
    }
```

9 Bypass Apache authentication

- */etc/apache2/conf-enabled/sso.conf*

```
<Perl>
    use lib '/etc/apache2/modperl';
</Perl>

PerlModule Apache2::Sso
PerlModule Apache2::noAuthz
PerlAddAuthzProvider  group Apache2::noAuthz
```

- *mdtx-paies/public_html/index/.htaccess*

```
SetHandler perl-script
PerlResponseHandler Apache2::Sso
PerlSetVar COLLECTION paies
PerlSetVar GROUP index

# login/password
Require group index
```

- */home/nroche/modperl/Apache2/noAuthz.pm*

```
package Apache2::noAuthz;

use Apache2::Const -compile => qw(AUTHZ_GRANTED);

sub handler {
    use strict;
    use warnings;

    my $r = shift;

    $r->log_error("->authz");
```

```
    return Apache2::Const::AUTHZ_GRANTED;
}

1;
```