

Maintain State with CGI

Perl and the World Wide Web

Doug Treder

UW Extension

Maintaining State

- Digest::MD5
- Pipelines
- Hidden fields
- Cookies
- Authentication
- Apache::Session

Digest::MD5

- The Digest modules can produce a *one-way hash* for a string
- Can slurp a whole file and digest it
- Or just a login and password
- Information is definitely lost

Digest::MD5

- Digest modules promise three things:
 - the same string should always produce the same digest
 - odds are astronomically small that two different strings produce the same digest
 - because of #2, it's very hard to figure out the original string from the digest

Digest::MD5

- MD5 is probably the quickest
- SHA1 is slightly better (though it's been supposedly cracked too)
- They're all still pretty tough to crack

Digest::MD5

- Producing a digest:

```
use Digest::MD5 qw(md5_hex);
```

```
my $password = "magic";
```

```
print md5_hex("magic");
```

```
2f3a4fcca6406e35bcf33e92dd93135
```

Checking a password

- Check a password without hardcoding it:

```
use Digest::MD5 qw(md5_hex);

my $encrypted_password =
    "2f3a4fcca6406e35bcf33e92dd93135";

my $password = $q->param('password');
if (md5_hex($password) eq $encrypted_password) {

    print "you're in !";
}
```

Login and Password

- Combination of login and password:

```
use Digest::MD5 qw(md5_hex);

my $encrypted =
    "7bbc129ed59a4c593c69a10418ecdee";

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

if (md5_hex("$login$password") eq $encrypted) {
    print "you're in !";
}
```

Pipelines

- A pipeline is a series of HTML forms, which remembers data from each form.
- You can use hidden fields to accomplish this.

Hidden fields

```
<form method=POST>  
Your dog's name?  
<input type=text name=dogname>  
<input type=submit>  
</form>
```

Hidden fields

```
<form method=post>  
Your dog's age?  
<input type=text name=dogage>  
<input type=hidden name=dogname value="Spoon">  
<input type=submit>  
</form>
```

Hidden fields with CGI

```
my $q = CGI->new();
print $q->header();
print $q->start_html("Maintaining State");
my ($name, $age) = ($q->param('dogname'),
    $q->param('dogage'));

if ($name && $age) {
    print "<h1>Dog $name is $age yrs old!</h1>";
    exit;
}
else {
    print $q->start_form();
    if ($name) {
        print "Your dog's age? ", $q->textfield(-name=>'dogage');
        print $q->hidden(-name=>'dogname');
    }
    else {
        print "Your dog's name? " , $q->textfield(-name=>'dogname');
    }
    print $q->end_form();
}
```

Cookies

- More lame jokes about cookies than any other HTTP artifact!
- A Netscape invention that has become standard in all browsers
- A cookie saves data on the browser's computer for retrieval later.

What's in a Cookie

- Name
- Value
- Domain
- Path
- Lifespan (expiration)



What good are cookies?

- An easy way to store state
- But cookies can only hold a small value
- You should limit yourself to a few cookies per website (some browsers have built-in limits).

Cookies and Security

- Cookies can be marked for return only on a SSL connection (we won't cover this)
- Still, browsers can sometimes be tricked into returning cookies to a rogue application.
- Don't store risky data in the user's cookies!

How to set cookies

- Use CGI::Cookie (and Apache::Cookie later with mod_perl)
- The cookie is part of the *header*, before any HTML is sent – and before redirects!

Setting cookies

```
$cookie1 = new CGI::Cookie  
  (-name=>'ID', -value=>123456,  
   -expires=> '+3M');  
  
$cookie2 = new CGI::Cookie  
  (-name=>'authentication',  
   -value=>'124523728312841');  
  
print $q->header  
  (-cookie=>[$cookie1, $cookie2]);
```

Retrieving Cookies

```
my %cookies =  
    CGI::Cookie->fetch();  
  
my $id = $cookies{ID};
```

Using cookies for authentication

- A naïve solution might be:

```
my $cookie = new CGI::Cookie  
    (-name=> 'auth', -value=>1);
```

- But this is easily faked.
- Neither should you store the password (even encrypted) in a cookie.

A better way

- Use three cookies (or a triple value cookie)
- One stores the username
- One stores a timestamp (to enable you to force expiration)

A better way

- The third stores a encrypted hash of the username, time, and a secret key.
- When you retrieve the cookie, you hash the username and time with the secret key and see if you get the same result.

Cookie authentication

- example in shell

Apache::Session

- Another way to save state is Apache::Session
- It's misnamed – it can be used to store arbitrary data against a “sessionID”.
- Not related to authentication!!! – just saves data.
- It's up to you to get the sessionID passed along either in URL or cookie.

Apache::Session

- Get Apache::Session installed. You don't need a database – DB_File is good enough.
- Put the session ID either on the end of the URL or in a cookie – or both (see book for example)

Apache::Session

- Starting a new session

```
tie %s, 'Apache::Session::DB_File', undef,  
    {FileName => '/tmp/sessions'};
```

Apache::Session

- Retrieving old session:

```
tie %s, 'Apache::Session::DB_File', $id,  
    {FileName => '/tmp/sessions'};  
  
# You can put anything you like in %s  
# It will still be there next time the CGI runs, too.
```