

Wie realisiere ich einen File-Upload mit CGI?

Häufig möchte man eine Datei des Users auf den eigenen Server hochladen. Oder man hat ein eigenes kleines CMS programmiert. Doch wie kommen jetzt die Dateien auf den Server, ohne dass ich einen FTP-Client benutze?

Auch für diese Aufgabe kann man Perl/CGI verwenden!

Beim Hochladen sollte man sich aber darüber im Klaren sein, dass man hierbei Benutzereingaben nicht einfach so hinnehmen sollte, da dies eine große Sicherheitslücke bedeuten würde – und wer will seinen Server schon einem Hacker überlassen? Ich nehme an keiner...

Also, was brauch ich für einen File-Upload?

Man benötigt einfach ein HTML-Formular und ein Perl-Skript. Im Formular muss der enctype „multipart/form-data“ eingetragen sein. Mit einem anderen enctype funktioniert es nicht. Dann benötigt das Formular noch ein Feld, in dem man eine Datei auswählen kann. Dafür gibt es

```
<input type="file" ...>
```

Als method muss „post“ eingetragen sein, da es mit der GET-Methode nicht klappt.

So sieht ein Formular für einen File-Upload aus:

```
<form action="my_script.cgi" method="post"
  enctype="multipart/form-data">
  <input type="file" name="datei"><br />
  <input type="submit" name="hochladen">
</form>
```

Und wie sieht das CGI-Skript aus?

Das Grundgerüst sollte schon bekannt sein. Möglichst CGI.pm benutzen. use strict und use warnings verwenden usw. Der Skriptumpf sieht also so aus:

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

use strict;
use warnings;
use CGI;
use CGI::Carp qw(fatalsToBrowser);

my $cgi = CGI->new();
print $cgi->header(-type => 'text/html');
my %params = $cgi->Vars();

print $cgi->start_html(-title => 'Mein erster File-Upload');
```

Mit CGI.pm kann man schon einen Filehandle bekommen. Dafür gibt es die Methode `upload()`.
Mit

```
my $filehandle = $cgi->upload('fieldname');
```

bekommt man den Filehandle.

Bevor wir jedoch anfangen die Datei einfach so hochzuladen, kümmern wir uns ein wenig um die Sicherheit des Skripts. Als erstes überlegen wir uns, welche Zeichen in einem Dateinamen erlaubt sein dürfen. Normalerweise sollte man davon ausgehen, dass die „normalen“ Buchstaben (A-Z), Zahlen, der Unterstrich (`_`), der Punkt (`.`) und das Minus (`-`) ausreichen. Die restlichen Zeichen schmeißen wir einfach raus:

```
my $filename = $params{fieldname};  
$filename =~ s/[^A-Za-z0-9_\.\-]//g;
```

Grundsätzlich sollte man Benutzereingaben nicht vertrauen. Sie sind eigentlich immer so zu bearbeiten, dass kein fremder Code eingeschmuggelt werden kann.

Ein weiterer Punkt der Sicherheit betrifft die maximale Größe einer hochzuladenden Datei. Die Maximalgröße kann man mit der Konstanten `$CGI::POST_MAX`. Für die Beschränkung auf 1 MB:

```
#!/usr/bin/perl  
  
use CGI;  
$CGI::POST_MAX = 1024 * 1024;
```

Hierbei dürfen die gesamten Daten, die mit `post` gesendet werden nicht größer als 1MB sein.

Die Methode

Um das Programm übersichtlicher zu machen und auch etwas für zukünftige Programme zu haben, schreiben wir den Code für den Upload in eine extra Methode, die wir `upload_file()` nennen. In dieser Methode wird der Pfad zur Zieldatei zusammengesetzt und die Datei hochgeladen.

Die Methode sieht folgendermaßen aus:

```
sub upload_file{
  my ($filename,$filehandle,$path) = @_;
  my $target = $path.'/'.$filename;
  if(-e $target){
    print „Zieldatei existiert schon!“;
    exit(0);
  }
  else{
    binmode $filehandle;
    open(TARGET,">$target") or die $!;
    binmode TARGET;
    my ($buffer);
    while(read $filehandle,$buffer,1024){
      print TARGET $buffer;
    }
    close TARGET;
  }
}
```

Ich habe immer Recht

Beim Upload muss darauf geachtet werden, dass die richtigen Rechte gesetzt sind. Dabei muss der Ordner, in den geschrieben wird, die Rechte 777 bzw 666 aufweisen.

Exkurs:

Die Rechte einer Datei oder eines Ordners werden in drei Dreier-Gruppen dargestellt. Die erste Gruppe ist das Recht des Besitzers/Erstellers, die zweite Gruppe die Rechte der Usergruppe, zu der der Besitzer gehört. Die letzte Dreier-Gruppe repräsentiert die Rechte, die alle haben.

Man kann die Rechte in Zahlen oder in Buchstaben angeben:

rwX => Lese-, Schreib- und Ausführrechte

rw- => nur Lese- und Schreibrecht

Die einzelnen Positionen haben eine Wertigkeit:

r => 2^2

w => 2^1

x => 2^0

Die Rechte 777 für den Ordner heißen also rwX rwX rwX, erste Gruppe $2^2 + 2^1 + 2^0 = 7$; genauso bei der zweiten und dritten Gruppe...

Der Ordner muss diese Weitläufigen Rechte haben, da bei einem File-Upload die Dateien von dem Standarduser des Webservers hochgeladen werden. Dies ist in der Regel der User mit der niedrigsten Wertung.

Eine Lösung des Problems wäre Apaches suEXEC, bei dem das CGI-Skript als ein anderer User auftritt. Dies halte ich persönlich für nicht so gut, aber es gibt auch andere Meinungen dazu.

Wie sieht jetzt das Skript endgültig aus?

So:

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

use strict;
use warnings;
use CGI;
use CGI::Carp qw(fatalsToBrowser);
$CGI::POST_MAX = 1024 * 1024;

my $cgi = CGI->new();
print $cgi->header(-type => 'text/html');
my %params = $cgi->Vars();
my $path = '../uploads';
my $filehandle = $cgi->upload('datei');
my $filename = $params{'datei'};
$filename = (split(/\\/,$filename))[-1];
$filename =~ s/[^A-Za-z0-9_\.\-]//g;

print $cgi->start_html(-title => 'Mein erster File-Upload');
upload_file($filename,$filehandle,$path);
print $cgi->end_html();

##
# Subroutines
##

sub upload_file{
    my ($filename,$filehandle,$path) = @_;
    my $target = $path.'/'.$filename;
    if(-e $target){
        print „Zieldatei existiert schon!";
        exit(0);
    }
    else{
        binmode $filehandle;
        open(TARGET,">$target") or die $!;
        binmode TARGET;
        my ($buffer);
        while(read $filehandle,$buffer,1024){
            print TARGET $buffer;
        }
        close TARGET;
        print "<h2>Ihre Datei wurde unter $target gespeichert</h2>";
    }
}
```