Skripte zur Arbeitserleichterung und ein Archiv-Modul

Gepackte Koffer

Jeder Programmierer nutzt für die tägliche Arbeit eine Sammlung persönlicher Skripte. Um vier aus dem eigenen Fundus geht es hier. Alle brauchen jedoch Perl-Module, die nicht auf jedem Rechner vorhanden sind. Das Perl Archive Toolkit löst die Abhängigkeit und packt Skripte und Module in ein handliches Archiv. Michael Schilli

Wer auf mehreren Rechnern arbeitet, kennt das Problem, dass Perl-Skripte einen Rattenschwanz an Modulen benötigen. Sie zu installieren erfordert eine CPAN-Konfiguration und kostet Zeit. Autrijus Tang hat mit dem CPAN-Modul PAR (Perl Archive Toolkit) eine Möglichkeit geschaffen, Skripte inklusive aller erforderlichen Module in Archive zu verpacken, ähnlich wie das unter Java mit ».jar«-Dateien funktioniert.

Da PAR aber nicht unbedingt auf jeder Maschine verfügbar ist, um ein Archiv zu entpacken, gibt es sogar die Möglichkeit, Executables zu erstellen. Sie enthalten den Interpreter, alle nötigen Module, Bibliotheken sowie die Skripte. So lassen sich auch Perl-Skripte ausführen, denen im Normalfall sämtliche Module fehlen würden.

Auch für Entwickler, die oft einen Werkzeugkoffer voller kleiner Skripte mit sich führen, ist PAR sehr hilfreich. Skripte, die bei der täglichen Arbeit nützlich sind und kleinere Jobs erledigen, verwenden in der Regel Module aus der schier unendlichen Sammlung des CPAN. Die folgenden vier Skripte sind Teil meiner eigenen Werkzeugsammlung.

Passwortschnüffler

Ein kleiner Base-64-(De-)Kodierer zum Beispiel ist immer dann nützlich, wenn man im Web mit Basic-Authentifikation arbeitet. Neulich saß ich nämlich vor einem Dialogfenster wie in Abbildung 1 und hatte das Passwort vergessen. Glücklicherweise besaß der Browser einen Passwortmanager, der die nötigen Felder automatisch ausgefüllt hat. Aber leider besteht das Passwortfeld nur aus Sternchen.

Doch ein zwischen Browser und Server geschalteter Proxy (beispielsweise der aus dem Artikel von [1]) lässt schnell erkennen, dass ein String wie »dGVzdDpzZWNyZXQ=« durch die Leitung fließt (siehe Abbildung 2). Dummerweise überträgt der Browser Passwort und Benutzername mit dem Base-64-Algorithmus kodiert. Die Dekodierung ist dank Perl aber sehr einfach, das Skript aus Listing 1 bringt den Klartext zum Vorschein:

```
$ b64.pl -d dGVzdDpzZWNyZXQ=
test:secret
```

Der Benutzername ist »test«, das Passwort lautet »secret«. Diese Methode zeigt auch, dass Basic-Auth nur wenig nützt, wenn jemand an der Leitung schnüffelt. Umgekehrt zeigt folgender Aufruf, dass »b64.pl« auch die Kodierung beherrscht:

```
$ b64.pl test:secret
dGVzdDpzZWNyZXQ=
```

Weitere nützliche Anwendungen gibt es viele: Base-64-Kodierung kommt immer dann zum Einsatz, wenn es darum geht, binäre Daten in ein anzeigbares Format umzuwandeln. Viele E-Mail-Programme kodieren beispielsweise binäre Attachments mit Base 64.

Prozent-Hex im URL-Wald

Auch in vielen URLs befinden sich kodierte Sequenzen. Wenn der Browser per GET-Request einen Parameter an den

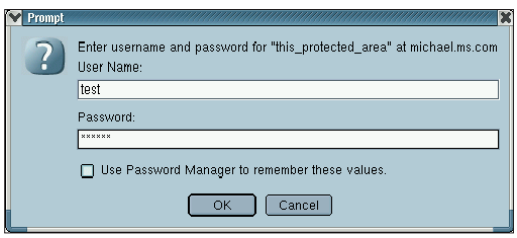

Abbildung 1: Ein typisches Browserfenster bei Basic-Authentifikation. Leider ist das automatisch vom Browser ausgefüllte Passwortfeld mit Sternchen verschleiert.

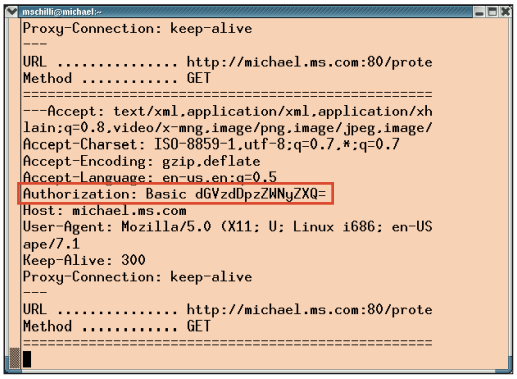

Abbildung 2: Ein Base-64-kodierter Passwortstring aus der Basic-Authentifikation (im Bild rot umrahmt) zwischen Webclient und Server lässt sich relativ einfach mit einem Proxy ausschnüffeln.

Server schickt, sieht das zum Beispiel folgendermaßen aus:

```
http://host.com/cgi/foo?p=a%20b%2Fc
```

Der Browser hat jedes Sonderzeichen in das Format »%XX« umgewandelt, wobei »XX« dem Ascii-Hexadezimalwert entspricht. Mit dem Skript in Listing 2 lässt sich der Hex-String leicht dekodieren:

```
$ urlcode.pl -d a%20b%2Fc
a b/c
```

Auch »urlcode.pl« beherrscht das Kodieren genauso wie das Dekodieren. Um einen String zu Testzwecken zu encoden, wandelt folgender Aufruf eine Zeichenkette ins URL-Format um:

```
$ urlcode.pl "a b/c"
a%20b%2Fc
```

Dank des CPAN-Moduls »URI::Escape« ist das Skript trivial, es testet lediglich mit »Getopt::Std«, ob »-d« auf der Kommandozeile angegeben wurde, und ruft dann entsprechend »uri_escape« oder »uri_unescape« aus dem Modul »URI::Escape« auf.

Atari-Hexdump

Binäre Dateien kleistern den Bildschirm mit Sonderzeichen voll und hängen manchmal sogar das Terminal auf, wenn Anwender sie mit »cat« anzeigen lassen. »less« ist besser, hilft aber auch nicht viel weiter, da es nicht angibt, wie lang bestimmte Byte-Sequenzen sind. Das CPAN-Modul »Data::Hexdumper« bietet eine Anzeige, wie sie schon vor 15 Jahren auf meinem Atari ST1040 zu sehen war: Links die Hexcodes in Gruppen von 16 Bytes und rechts die Übersetzung in lesbare Zeichen – falls das möglich ist. Abbildung 3 zeigt, wie »hd.pl« aus Listing 3 seinen eigenen Quellcode als Hexdump auf den Schirm schreibt.

Das Programm »hd.pl« ist zwar nur eine schon peinlich triviale Adaption der »Data::Hexdumper«-Manualseite – aber selbst ein so einfaches Werkzeug spart oft viel Zeit.

Perl-Switcheroo

Wer viel mit Perl arbeitet, stets mit zitternden Fingern die neueste Distribution ausprobiert oder nur schnell feststellen will, ob ein Modul auch mit Antik-Versionen wie 5.00503 funktioniert, der braucht die Möglichkeit, schnell zwischen verschiedenen Installationen hin und her zu springen. Dazu ist es ratsam, Perl-Distributionen nicht mehr unter »/usr« zu installieren, sondern in einem dedizierten Verzeichnis, etwa unterhalb des Homeverzeichnisses. Ein Configure-Aufruf für Perl 5.8.4 sieht dann so aus:

```
./Configure -D prefix=$HOME/➥
perl-installs/perl-5.8.4 -d
```

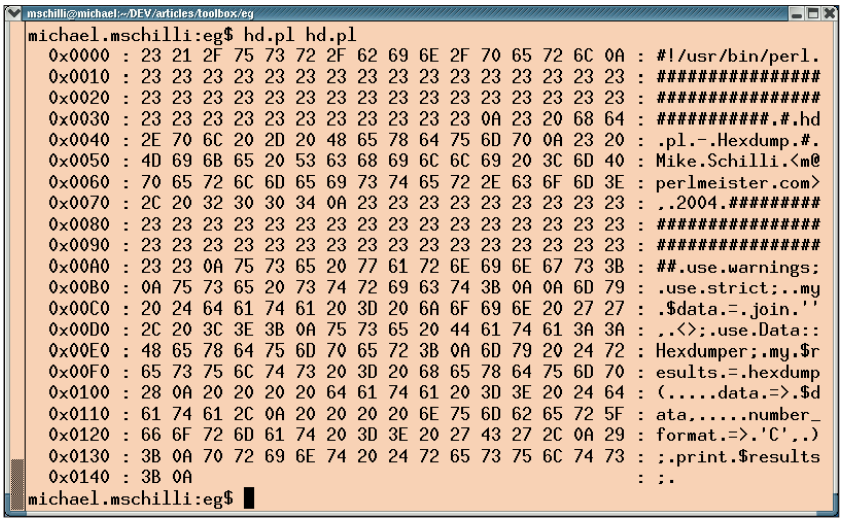

Abbildung 3: Ist es erforderlich, binäre Daten anzeigen zu lassen, hilft das Perl-Modul »Data::Hexdumper«. Hier zeigt das Skript »hd.pl« mit Hilfe des Moduls seinen eigenen Quellcode an.

Listing 1: »b64.pl«

```
01 #!/usr/bin/perl
02 ##########################################
03 # b64.pl - Encode and decode Base64
04 # Mike Schilli <m@perlmeister.com>, 2004
05 ##########################################
06 use warnings;
07 use strict;
08
09 use Getopt::Std;
10 use MIME::Base64;
11
12 getopts "d", \my %opts;
13
14 die "usage: $0 [-d] string" unless
15     defined $ARGV[0];
16
17 if($opts{d}) {
18     print decode_base64($ARGV[0]), "\n";
19 } else {
20     print encode_base64($ARGV[0]);
21 }
```

Listing 2: »urlcode.pl«

```
01 #!/usr/bin/perl
02 ##########################################
03 # urlcode.pl - URL en/decode
04 # Mike Schilli <m@perlmeister.com>, 2004
05 ##########################################
06 use warnings;
07 use strict;
08
09 use Getopt::Std;
10 use URI::Escape;
11
12 getopts "d", \my %opts;
13
14 die "usage: $0 [-d] string" unless
15     defined $ARGV[0];
16
17 if($opts{d}) {
18     print uri_unescape($ARGV[0]), "\n";
19 } else {
20     print uri_escape($ARGV[0]), "\n";
21 }
```

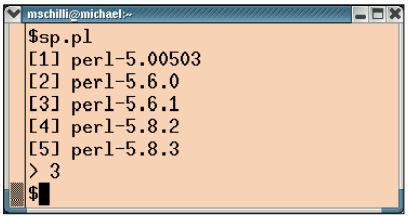

Abbildung 4: Wer schnell zwischen verschiedenen installierten Perl-Versionen wechseln muss, benutzt einfach ein Skript, das die nötige Arbeit im Hintergrund erledigt.

Der Aufruf von »make install« installiert die Distribution unter dem angegebenen Verzeichnis. Um schnell zwischen den Versionen zu wechseln, ist ein symbolischer Link »perl-current« in »perl-installs« anzulegen, der auf die gewünschte Version zeigt. Anschließend erstellt der Programmierer noch einen Link von »/usr/bin/perl« auf »$HOME/perl-installs/perl-current/bin/perl«.

Listing 3: »hd.pl«

```
01 #!/usr/bin/perl
02 ##########################################
03 # hd.pl - Hexdump
04 # Mike Schilli <m@perlmeister.com>, 2004
05 ##########################################
06 use warnings;
07 use strict;
08
09 my $data = join '', <>;
10 use Data::Hexdumper;
11 my $results = hexdump(
12     data => $data,
13     number_format => 'C',
14 );
15 print $results;
```

Listing 4: »sp.pl«

```
01 #!/usr/bin/perl
02 ##########################################
03 # sp.pl - Select a perl installation
04 # Mike Schilli <m@perlmeister.com>, 2004
05 ##########################################
06 use strict;
07 use warnings;
08
09 use File::Basename qw(basename);
10
11 my $PERL_HOME = "$ENV{HOME}/perl-installs";
12
13 my(@versions, $count);
14
15 for (<$PERL_HOME/perl-*>) {
16     next if -l or ! -d;
17     push @versions, basename($_);
18 }
19
20 foreach my $v (@versions) {
21     print "[", ++$count, "] $v\n";
22 }
23
24 $| = 1;
25 print "> ";
26 my $number = <>;
27 chomp $number;
28
29 die "Invalid choice" unless
30   exists $versions[$number-1];
31
32 unlink("$PERL_HOME/perl-current") or
33     warn "unlink failed ($!)";
34 symlink("$PERL_HOME/$versions[$number-1]",
35         "$PERL_HOME/perl-current") or
36     die "symlink failed ($!)";
```

So greifen Skripte, in deren Shebang-Zeile »#!/usr/bin/perl« steht, auf die aktuell gewählte Perl-Installation zu. Auch andere aus der Distribution genutzte Programme beziehungsweise Skripte wie etwa »perldoc« sind entsprechend umzubiegen.

Nach diesen Vorbereitungen kommt das Skript »sp.pl« (für Switch Perl) in Listing 4 zum Einsatz. Wie Abbildung 4 zeigt, bietet es eine Auswahl von unter »$HOME/perl-installs« installierten Distributionen an und setzt daraufhin den symbolischen Link »perl-current« entsprechend um. Praktisch!

Koffer packen mit PAR

Nun verlangen diese vier vorgestellten Werkzeuge allerdings einige Zusatzmodule auf der Zielmaschine: »b64.pl« benutzt »MIME::Base64« und »urlcode.pl« braucht »URI::Escape«. Beide sind nicht in der Standard-Perl-Distribution enthalten und darum mit einiger Wahrscheinlichkeit nicht auf allen Maschinen installiert, für die sie bestimmt sind.

Um alle vier vorgestellten Skripte in ein Archiv zu packen, genügt es, das Programm »pp« aus der PAR-Distribution vom CPAN aufzurufen:

```
pp --output=toolbox.exe b64.pl ➥
urlcode.pl hd.pl sp.pl
```

Der Aufruf erzeugt das Binary »toolbox.exe«, das alle vier Skripte und die erforderlichen Zusatzmodule enthält. Trotz der Endung ».exe« hat das Binary immer das Format des Betriebssystems, auf dem der Programmierer »pp« ausgeführt hat. Fährt die Zielmaschine dieselbe Plattform, ist eine Installation des Werkzeugkoffers einfach. Die folgenden Zeilen installieren alles Nötige im Verzeichnis »~/bin/toolbox«:

```
mkdir ~/bin/toolbox
cp toolbox.exe ~/bin/toolbox
cd ~/bin/toolbox
for i in b64 urlcode hd sp ; do
    ln -s toolbox.exe $i
done
export PATH=$PATH:~/bin/toolbox
```

Die symbolischen Links zeigen alle auf »toolbox.exe« und PAR findet selbst heraus, was gemeint ist, wenn der Benutzer zum Beispiel »b64« aufruft. Es extrahiert »b64.pl« aus dem Archiv, lädt die nötigen Zusatzmodule ebenfalls aus dem Archiv und führt das Skript aus.

PAR unterstützt sogar mehrere Plattformen gleichzeitig, dafür sind die Werkzeuge mit der Option »--multiarch« nacheinander auf jedem Zielsystem in die Kiste zu packen. Natürlich funktioniert das nicht als ausführbare Datei, da jedes Betriebssystem ein anderes Format benutzt. Der Anwender muss für Multiarch das PAR-Modul auf seinem Rechner installiert haben. Das Tutorial, das PAR als »PAR::Tutorial« beiliegt, gibt weitere Tipps zur Benutzung.

Einen Stolperstein gilt es allerdings noch zu beachten: Programmierer sollten das PAR-Archiv stets mit einer möglichst alten Maschine bauen, da es sonst auf alten Maschinen Probleme mit der Libc gibt. *(mwe)*

Infos

[1] Michael Schilli, „Loggender Proxy“: Linux-Magazin 04/00, [http://www.linux-magazin.de/Artikel/ausgabe/2000/04/Proxy/proxy.html]

Der Autor

Michael Schilli arbeitet als Software-Entwickler für AOL/Netscape in Mountain View, Kalifornien. Er hat „Goto Perl 5“ (deutsch) und „Perl Power“ (englisch) für Addison-Wesley geschrieben und ist unter [mschilli@perlmeister.com] zu erreichen. Seine Homepage heißt [http://perlmeister.com].