Script CGI

Parleremo della CGI (Common Gateway Interface), ma prima di tutto devi sapere alcune cose sulle problematiche client/server. In altre parole devi sapere come i server ed i client si parlano l' un l' altro. Vediamo...

Conversazioni tra un client ed un server

Quando 'clicchi' su un link contenuto all' interno di una pagina web, forse non sai che il tuo browser inizia a chiaccherare con il server puntato da quel link. Un client e' qualsiasi programma software che puo' chiedere qualcosa ad un server (spesso anche lo stesso computer dove sta girando il client e' considerato client). Percio' il web browser e' un client. Un server e' qualsiasi programma software che gira su un computer collegato ad una rete in grado di rispondere alle richieste fatte dai client (spesso anche lo stesso computer dove gira il server e' considerato un server). Quando digiti un URL (Uniform Resource Locator) nel campo location del tuo browser, o quando clicchi su un link (link od URL sono la stessa cosa), stai puntando ad una certa pagina web contenuta all' interno di un dato computer. Un server web e' un programma software che gira ininterrottamente su una macchina, in attesa di richieste da soddisfare. Un server e' chiamato anche demone. Quando il server riceve una richiesta, cerca di soddisfarla. In altre parole, quando clicchi su un link, stai chiedendo (beh, e' il tuo browser che effettua la richiesta) una pagina web (documento). Il server riceve la tua richiesta e ricerca quella pagina, e, quando la trova, la consegna al browser web. I client ed i server parlano usando un particolare linguaggio: HTTP. HTTP sta per HyperText Transfer Protocol. Vediamo quest' URL: http://www.anyserver.com/anydirectory/anypage.html. Ci sono 4 parti:

1. Il nome del metodo (o servizio)
2. Il nome del server
3. Un path opzionale
4. Il nome della pagina web

Il nome del metodo puo' essere HTTP (web), FTP (FTP), NEWS (newsgroups), MAILTO (SMTP, Simple Mail Transfer Protocol, cioe' email) e cosi' via. Il nome del server indica il nome del computer dove si trova la pagina. Il path opzionale indica la directoty dove e' contenuta la pagina. Ed infine c'e' il nome della pagina.

Quando clicchi su http://www.anyserver.com/anydirectory/anypage.html, prima di tutto il tuo client (il browser web) attiva una connessione tra il tuo computer ed il server www.anyserver.com, richiedendo una pagina web di nome anypage.html. A questo punto il server riceve la richiesta e ricerca la pagina. Se il server la trova, la consegna al client, altrimenti invia un messaggio di errore molto noto: '404 not found'. Infine il server chiude la connessione.

L' HTTP e' un protocollo stateless, che significa che ogni richiesta fatta dal tuo client al server, richiede una nuova connessione. Ogni oggetto contenuto all' interno di una pagina web richiede una nuova connessione (a differenza di altri protocolli come FTP o Telnet, dove il client stabilisce una sola connessione e puo' effettuare parecchie richieste durante quella connessione). Percio' quando scarichi una pagina web che contiene parecchie immagini per esempio, il tuo client deve stabilire una nuova connessione per ogni oggetto che trova leggendo la pagina. Bene, durante una nuova richiesta il server non e' in grado di sapere nulla della precedente richiesta, e, difatti, non sa neppure chi ha fatto la precedente richiesta. Il nuovo protocollo HTTP (HTTP−NG o HTTP Next Generation, permette una

connessione per piu' richieste, cosi', per esempio, puoi leggere una pagina web stabilendo solo una connessione. Cosi' una pagina web e' scaricata piu' velocemente). Dunque il server web e' solo un programma che gira ininterrottamente aspettando delle richieste. Quando una richiesta viene soddisfatta e la connessione viene chiusa, il server e' nuovamente pronto per nuove richieste. In altre parole il server web e' una specie di cameriere che obbedisce agli ordini dei client. Comunque il tuo browser puo' richiedere documenti HTML, immagini, suoni, video o...script. Solo che uno script non viene ricevuto ma viene ESEGUITO. Ed ora vediamo cosa sono questi script CGI. Uno script e' un programma che funge da interfaccia tra un client, un server web, il sistema operativo, le periferiche hardware o anche altri server. Per esempio, supponi che tu voglia effettuare ricerche all' interno di un database. Supponi di volerle fare utilizzando una interfaccia web (una form per esempio). Bene, il server che offre tale servizio, ha bisogno sicuramente di uno script CGI, infatti deve ricevere la tua richiesta (che di solito e' scritta usando il linguaggio umano) e tradurla secondo le regole del linguaggio di quel database. Pensa ad un motore di ricerca come Altavista per esempio. Lo script CGI riceve la tua interrogazione e la traduce secondo le regole del linguaggio del database di Altavista e ricerca il documento che hai richiesto. Quando trova qualcosa lo deve confezionare per mostrarlo all' interno di una pagina HTML (in altre parole, lo script CGI deve costruire una pagina web 'al volo' per mostrare i risultati della tua interrogazione).

Linguaggi usati

Puoi usare qualsiasi linguaggio di programmazione per costruire uno script CGI. Beh, ovviamente il linguaggio di programmazione che scegli deve essere supportato dal server. Per esempio, se il tuo server sta girando su un sistema Unix, potresti usare il linguaggio della shell Bourne, o potresti usare il linguaggio C. Se il tuo server sta girando su un Macintosh devi usare il linguaggio Applescript, e cosi' via. Comunque il linguaggio di programmazione piu' usato per scrivere degli script CGI e' sicuramente il PERL (PERL sta per Practical Extraction and Reporting Language).

Breve corso sulla CGI

Home

Form e script CGI

Di solito gli script CGI e le form vivono insieme. Infatti di solito gli script ricevono dei dati dagli utenti, elaborano le informazioni ed infine mostrano i risultati agli utenti (di solito costruendo pagine web 'al volo'). Comunque potresti eseguire uno script CGI semplicemente cliccando su un link (cio' si verifica quando il link punta ad uno script CGI invece che ad un documento HTML). Percio' per scrivere script CGI devi sapere come funzionano le form. Una form e' solo una pagina web che contiene dei campi di input (per digitarci delle informazioni dentro) ed uno o piu' bottoni (per 'sottomettere' una form ed inviare i dati). Inoltre devi specificare lo script CGI che trattera' quelle informazioni. Ma cominciamo con l' osservare una semplice form. Ecco il codice HTML:

```
<html><head><title>This is a simple form</title></head>
<body>
<FORM METHOD="GET" ACTION="http://www.anyserver.com/cgi−bin/formsname">
<p>Type something here: <INPUT NAME="nameoffield"></p>
<p>INPUT TYPE="SUBMIT"></p>
</FORM>
</body></html>
```

<FORM></FORM>

Con questi tag stai dicendo al browser web (il tuo client): 'Ehi browser! Questa e' una form e devi leggerla per inviare i dati al server specificato'.

METHOD="GET"

Tramite l' opzione 'METHOD', stai dicendo al browser web COME deve inviare i dati. Puoi specificare 2 modi: GET o POST. Se specifichi GET come metodo, i dati saranno inviati *accodandoli alla fine dell' URL*, altrimenti, usando il metodo POST, i dati saranno inviati *come un distinto oggetto HTML* al server web (vedremo piu' tardi la reale differenza tra questi metodi). Attenzione: non ti confondere con i metodi HTML e quelli HTTP! Qui sto parlando di METODI HTML (come inviare i dati). Il metodo GET HTTP per esempio (contenuto all' interno della 'chiaccherata' tra un client ed un server), e' una cosa completamente differente (infatti significa che il server web deve inviare un dato documento).

ACTION="http://www.anyserver.com/cgi−bin/scriptname"

Con l' opzione 'ACTION', stai dicendo al server web COSA deve fare, o meglio, QUALE SCRIPT deve girare e DOVE si trova detto script. Percio' anyserver eseguira' lo script scriptname, che si trova all' interno della directory cgi−bin. Cgi−bin non e' solo una semplice directory: e' una directory particolare. Infatti puoi configurare un server web per fargli sapere dove puo' trovare gli script CGI. Di solito gli script CGI si trovano all' interno della directory cgi−bin. Comunque il server web puo' essere configurato in modo da eseguire tutti i file con estensione 'cgi' (in questo caso il file 'script.cgi' per esempio, puo' essere eseguito come script CGI dal server web).

INPUT NAME="nameoffield"

Questo e' un campo di input, e qui gli utenti possono digitare le informazioni richieste. Lo script CGI

guardera' all' interno di una variabile (in questo esempio il nome di tale variabile e' 'nameoffield').

INPUT TYPE="SUBMIT"

Beh, questo e' solo un bottone per sottomettere la form. Una volta compilata la form, gli utenti possono inviare i dati premendo qusto bottone. Se specifichi GET come metodo, i dati contenuti all' interno dei campi INPUT vengono accodati alla fine dell' URL, dopo il simbolo '?'. Inoltre tutti i campi e i loro rispettivi nomi sono separati dal simbolo '='. Nel precedente esempio, quando premi il bottone submit, il browser web compone questo URL:

http://www.anyserver.com/cgi−bin/scriptname?nameoffield=*something*

Se hai piu' campi di input, il browser web mette il simbolo '&' fra ogni campo. Per esempio, suponi di avere 2 campi di input chiamati 'field1' e 'field2', bene l' URL sara':

http://www.anyserver.com/cgi−bin/scriptname?field1=*something*&field2=*something*

Dove '*something*' e' qualsiasi cosa l' utente ha digitato all' interno del campo. Se l' utente digita 2 o piu' parole all' interno di un campo di input, il browser mette un simbolo '+' tra ognuno di loro. In altre parole sostituisce gli spazi col simbolo '+'. Percio' supponi di avere 2 campi: 'yourname' e 'yourage'. Poi supponi che l' utente compili la form cosi':

yourname = Robert Clark

yourage = 99

Bene, l' URL sara':

http://www.anyserver.com/cgi−bin/scriptname?yourname=*Robert*+*Clark*&yourage=*99*

Tutti i caratteri speciali come =, &, #, £, %, / e cosi' via, vengono preceduti da '%'. cosi' nell' esempio precedente l' utente potrebbe scrivere:

yourname = Rober Clark & Company

E l' URL dopo il nome dello script diventera':

yourname=*Robert*+*Clark*+%&+*Company*&yourage=*99*

Attenzione: quando l' utente preme il bottone submit, il browser web costruira' l' URL, e le variabili saranno inviate accodate alla fine dell' URL stesso. Quando il server web riceve i dati, le variabili vengono memorizzate all' interno di una particolare variabile d' ambiente chiamata PATH_INFO. Lo script CGI puo' leggere quella variabile, ma deve decodificarla, perche' contiene tutti gli indesiderabili simboli come &, +, = e cosi' via. Vedremo le variabili d' ambiente successivamente.

Sezioniamo il piu' semplice script: Salve Rete!

Questo e' uno script molto semplice:

```
#!/bin/sh

#Filename: Salve.sh.cgi
echo "Content−type: text/html"
echo

echo "<HTML><HEAD><TITLE>Salve Rete!</TITLE></HEAD>"

#HTML body

ECHO "<BODY><H1>Salve Rete!</H1></BODY></HTML>"
```

Spiegazione

Prima di tutto devi sapere che di solito uno script CGI:

1. Legge dei dati in input (di solito tramite una form)
2. Fa qualcosa
3. Invia il risultato dell' elaborazione al client (il browser web)

Ok, ora vediamo la prima riga: #!/bin/sh

Il primo simbolo (#) e' un carattere speciale usato per definire una riga di commento, e l'interprete dello script non la eseguira'. Comunque questa e' una riga molto speciale, e devi scriverla all' interno di TUTTI gli script CGI. Se ometti questa riga, non potrai eseguire lo script. Infatti questa riga specifica al server web QUALE interprete deve usare e DOVE trovarlo. In questo esempio stai specificando che vuoi usare il linguaggio della Bourne Shell (scritto da Steve Bourne) tramite 'sh' e che l' interprete del linguaggio si trova nella directory: '/bin' (questa e' una directory speciale dove sono contenuti parecchi comandi Unix). Bene, ho appena detto che '#' definisce un commento. E' giusto, ma quando usi il simboloI '!' non e' piu' vero...Percio' tramite la riga '#!/bin/sh' stai dicendo: 'Ehi, server! Usa l' interprete della shell Bourne per eseguire questo script! Puoi trovarlo in /bin...'.

Bene, la seconda riga e' solo un commento, ma la terza riga:
echo "Content−type: text/html"

STAMPA la riga 'Content−type: text/html' sullo stdout (standard output). Stiamo parlando di sistemi Unix e qui c' e' un file speciale chiamato 'standard output' che di solito indica lo schermo. Beh, non e' proprio cosi', infatti un server web e' composto da: sistema operativo (di solito Unix), server web (un programma software) ed eventualmente alcuni script CGI. Lo script CGI stampa i dati sullo standard output (lo schermo sui sistemi Unix) ma il server web legge l' output dello script. Non devi necessariamente conoscere i dettagli del sistema operativo Unix per costruire un semplice script CGI, devi solo sapere che devi seguire alcune regole (infatti la CGI e' solo un insieme di regole...). Ad ogni modo lo script CGI dice al browser che sta per inviare un spemlice file di testo, e che quel file e' un documento HTML tramite l' intestazione Content−type (se non lo specifichi il browser web leggera' il

file come un semplice file ASCII e tu non vedrai la pagina web). Ecco aluni cormati di file abbastanza comuni:

HTML	text/html
Text	text/plain
GIF	image/gif
JPEG	image(jpeg
PostScript	application/postscript
MPEG	video/mpeg

I formati specificate qui sopra sono chiamati anche tipi MIME. MIME sta per Multipurpose Internet Mail Extension e venne originariamente sviluppato per espandere le possibilita' offerte dal sistema di posta elettronica. Ci sono 3 tipi di intestazione: Content−type, Location e Status. A volte non hai la necessita' di dover scrivere delle pagine HTML 'al volo', perche' puoi inviare al browser web una pagina web scritta precedentemente. Cosi' puoi specificare al server web dove puo' trovare la pagina. In questo caso puoi usare l' intestazione 'location'. Per esempio: 'Location: ../pages/mypage.html'. Ovviamente, se usi questa intestazione, non devi usare l' intestazione 'content−type'. A volte non devi inviare proprio nulla al browser, percio' puoi inviare una riga come questa: 'Status: 204 No Response'. In questo caso il browser web sapra' che non deve aspettarsi ulteriori risposte dal server.

La quarta riga e' necessaria per stampare un paio di caratteri newline. Questa e' una regola: dopo il content−type devi stampare 2 newline (echo stampa 1 newline, cosi' la riga precedente − la terza − scrive il primo newline, e questa riga − la quarta − stampa il secondo newline). Questo perche' il server web usa 2 newline per rilevare la fine dell' intestazione (che puo' essere content−type, location o status).

Infine ci sono i tag HTML che definiscono la pagina web che visualizza: Salve Rete!

Quello che segue e' lo stesso script scritto in PERL:

```
#!/usr/bin/perl
#Filename: salve.pl
print "Content−type: text/html"

#HTML body

ECHO "<BODY><H1>Salve Rete!</H1></BODY></HTML>"
```

<<<indietro Home CGI avanti >>>

Variabili d' ambiente

SERVER_NAME	Il nome dell' host (o l' indirizzo IP) dove sta girando lo script
SERVER_SOFTWARE	Il software del server usato. Esempi: CERN/3.0 o NCSA/1.3
GATEWAY_INTERFACE	La versione CGI che sta girando sul server
SERVER_PROTOCOL	La versione del protocollo HTTP usato. Dovrebbe essere CGI/1.1
SERVER_PORT	La porta TCP alla quale il server e' collegato. Di solito questo valore e' 80
REQUEST_METHOD	Il metodo usato: GET o POST
HTTP_ACCEPT	L' elenco dei Content−type che il browser e' in grado di gestire
HTTP_USER_AGENT	Il nome del browser che ha inviato le informazioni. Questa variabile di solito contiene il nome del browser, la sua versione ed altre informazioni come la piattaforma usata
HTTP_REFERER	L' URL del documento che contiene la form
PATH_INFO	L' URL ed informazioni aggiuntive inviate dal browser quando viene usato il metodo GET (vedi Sezioniamo un semplice script CGI: Salve Rete! per ulteriori informazioni)
PATH_TRANSLATED	Il path reale del sistema contenuto all' interno della variabile PATH_INFO
SCRIPT_NAME	Il nome dello script ed il path
QUERY_STRING	Informa<ioni inviate tramite il metodo GET. In altre parole tutte le informazioni inviate dopo il simbolo '?' all' interno dell' URL
REMOTE_HOST	Il nome dell' host che ha effettuato la richiesta
REMOTE_ADDR	L' indirizzo dell' host che ha effettuato la richiesta
REMOTE_USER	Il nome dell' utente che ha inviato la richiesta (quando e' disponibile una forma di autenticazione)
REMOTE_IDENT	Il nome del server che ha inviato la richiesta (quando e' disponibile il protocollo ident)
CONTENT_TYPE	Quando e' usato il metodo POST, il suo valore e': 'application/x−www−form−urlencoded'. Quando viene inviato un file, il suo valore e': 'multipart/form−data'
CONTENT_LENGTH	Quando e' usato il metodo POST, il suo valore definisce la dimensione del canale di input. In altre parole il numero di bytes inviati

Un semplice script CGI per mostrare le variabili d' ambiente

```
#!/usr/bin/perl
# Filename: vars.perl.cgi
# Send the MIME header
print "Content−type: text/plain\n\n";

# CGI standard variables
print "GATEWAY_INTERFACE = $ENV{'GATEWAY_INTERFACE'}\n";
print "REQUEST_METHOD = $ENV{'REQUEST_METHOD'}\n";
print "SCRIPT_NAME = $ENV{'SCRIPT_NAME'}\n";
print "QUERY_STRING = $ENV{'QUERY_STRING'}\n";
```

```
print "SERVER_SOFTWARE =  $ENV{'SERVER_SOFTWARE'}\n";
print "SERVER_NAME = $ENV{'SERVER NAME'}\n";
print "SERVER_PROTOCOL =  $ENV{'SERVER_PROTOCOL'}\n";
print "SERVER_PORT = $ENV{'SERVER_PORT'}\n";
print "HTTP_USER_AGENT =  $ENV{'HTTP_USER_AGENT'}\n";
print "HTTP_ACCEPT = $ENV{'HTTP_ACCEPT'}\n";
print "PATH_INFO = $ENV{'PATH_INFO'}\n";
print "PATH_TRANSLATED =  $ENVI{'PATH_TRANSLATED'}\n";
print "REMOTE_HOST = $ENV{'REMOTE HOST'}\n";
print "REMOTE_ADDR =$ENV{'REMOTE_ADDR'}\n";
print "REMOTE_USER = $ENV{'REMOTE_USER'}\n";
print "REMOTE_IDENT = $ENV{'REMOTE_IDEN'}\n";
print "AUTH_TYPE = $ENV{'AUTH_TYPE'}\n";
print "CONTENT_TYPE = $ENV{'CONTENT_TYPE'}\n";
print "CONTENT_LENGTH =  $ENV{'CONTENT_LENGTH'}\n";
```

<<<indietro Home CGI avanti >>>

Passo dopo passo: installare uno script CGI

Prima di tutto devi avere un accesso ad un server. Poi devi essere autorizzato ad installare degli script CGI. Dopo aver scritto il tuo script CGI devi:

1. Verificare dove si trova l' interprete PERL (se scrivi uno script in PERL) e dove si trova il programma SENDMAIL sul tuo server
2. Chiedere all' amministratore di sistema dove vanno inseriti gli script CGI (per esempio all' interno della directory 'cgi−bin')
3. Trasferire lo script sul server
4. Cambiare i permessi di accesso sui file tramite il comando 'chmod' o tramite il tuo client FTP

Passo 1: dov' e' l' interprete PERL?

Se scrivi uno script nel linguaggio della shell Bourne, non devi preoccuparti del PERL. Ma se scrivi uno script in PERL, devi sapere dove si trova l' interprete. Di solito l' interprete PERL si trova all' interno della directory '/usr/bin' directory (sui sitemi Unix) ma i tuo sistema potrebbe avere una configurazione diversa. Percio' devi chiedere al tua amministratore di sistema. Inoltre devi sapre dove si trova il programma SENDMAIL.

Passo 2: dove devi mettere il tuo script?

Dipende dalla configurazione del server, infatti potresti dover semplicemente scrivere un file con estensione 'cgi' o potresti dover mettere lo script all' interno di una directory particolare. Di solito la directory designata e': 'cgi−bin'.

Passo 3: trasferire lo script sul server

Ok, ora hai tutte le informazioni necessarie e devi trasferire lo script sul server. Prima regola: devi trasferire lo script usando la modalita' di trasferimento ASCII. Potresti avere dei problemi se usi la modalita' BINARIA. Per esempio: il carattere newline e' un simbolo codificato in ASCII, percio' il tuo client FTP deve trasferire lo script usando il sistema di codifica ASCII. Se usi la modalita' binaria (BINARY), il carattere newline non verrebbe considerato come carattere ASCII e verrebbe tradotto erroneamente.

Passo 4: cambiare i permessi sui file

Devi cambiare i permessi sui file, perche' il tuo script deve essere eseguibile. Sui sistemi Unix il comando 'chmod' cambia i permessi sui file. I permessi sono: 'r' (lettura), 'w' (scrittura) ed 'x' (esecuzione). Percio' devi usare il flag 'x'. Puoi abilitare le operazioni usando il simbolo '+' e disabilitarle usando il simbolo '−'. Siccome devi abilitare l' esecuzione, il comando 'chmod' sara': 'chmod +x scryptname.cgi' (in questo caso stai abilitando l' esecuzione di scriptname.cgi a tutti gli utenti. Comunque se il tuo server sta girando usando il tuo UID (User IDentifier) devi cambiare il comando chmod cosi':
'chmod u +x scriptname.cgi'. Infatti nel primo esempio stai supponendo che il tuo server stia girando come 'nobody' (cioe' 'nessuno', percio' tutti gli utenti dovrebbero essere abilitati ad eseguire lo script), mentre nel secondo esempio il tuo server sta girando usando il tuo UID.

<<<indietro Home CGI avanti >>>

Il mio script non funziona: qual e' il problema?

Prima di tutto devi sapere che il tuo server ha inviato un messaggio di errore allo stantard error (di solito lo schermo), ma se non riesci a vedere questi messaggi, devi controllare il log del server.

FAQ

- D. Perche' il server risponde: '404 not found'?
- R. Puo' essere che l' URL specificato non sia corretto
- D. Perche' il server risponde: '403 access forbidden'?
- R. Il permesso di esecuzione del file non e' stato concesso. Devi cambiare i permessi di accesso del file. Infatti uno script CGI e' un file eseguibile, percio' il tuo script deve essere eseguibile. Sui sistemi Unix devi usare il comando 'chmod' per cambiare i permessi di accesso al file. Il tuo server puo' girare come 'nobody' (nessuno) o usando il tuo UID (User IDentifier). nel primo caso puoi impostare: 'chmod +x yourscript.cgi'. Nel secondo caso puoi impostare: 'chmod u +x yourscript.cgi'
- D. Perche' il server risponde: 'malformed header from script'?
- R. Hai inserito l' intestazione 'Content−type'? Questa riga e' necessaria
- D. Perche' il server risponde: '500 server error'?
- R. Beh, ci sono molte ragioni per ricevere questo messaggio, perche' questo e' un messaggio di errore generico, percio' puo' essere che:
    - I permessi di esecuzione non sono stati impostati correttamente (vedi la domanda precedente)
    - riga interpretata erroneamente: hai dimenticato questa prima riga indispensabile: #!/usr/bin/perl. Comunque il tuo path potrebbe essere diverso, percio' devi sapere dove si trova l' interprete PERL
    - Errore di sintassi nello script: controlla lo standard error oppure il log del server
    - Non trovato il file richiesto: hai richiesto una libreria tramite 'require' o 'use', ma la libreria non puo essere trovata
    - Errore nello script in esecuzione: lo script ha causato un errore in esecuzione (per esempio un file non trovato una divisione per zero e cosi' via)
    - Intestazione HTTP invalida: la prima riga di output inviata dallo script deve sempre essere una intestazione HTTP valida, di solito: Content−type: text/html
- D. Perche' il browser dice: 'Document contains no data'?
- R. Significa: lo script ha inviato una valida intestazione, ma non ha inviato ulteriori informazioni. Cause possibili:
    - Errore nello script
    - Server timeout
    - nessun output inviato dal tuo script
- D. Perche' il browser visualizza il testo dello script invece di eseguirlo?
- R. Puo' darsi che tu abbia installato lo script nel posto sbagliato oppure il nome che gli hai dato non e' corretto. Forse non hai installato lo script all' interno della directory cgi−bin, o forse il nome dello script non contiene il suffisso 'cgi'. In questo caso infatti lo script sara' ritornato come un normale file di testo
- D. Perche' il browser mi chiede di salvare lo script quando tendo di eseguirlo?
- R. Puo' darsi che lo script abbia inviato una errata intestazione 'Content−type'. Infatti in questo caso il tuo browser non sara' capace di riconoscere il formato corretto (tipo 'sconosciuto') e ti chiedera' se vuoi salvarlo

<<<indietro Home CGI avanti >>>

Codici di stato HTTP

- Ok 200 − La richiesta ha avuto successo
- Created 201 − La richiesta ha avuto successo ed il risultato e' la creazione di una nuova risorsa
- Accepted 202 − La richiesta e' stata accettata, ma l' elaborazione non e' stata completata
- Partial Information 203
- No Response 204 − Il server ha ricevuto la richiesta ma non ha ritornato indietro alcuna informazione
- Moved 301 − Spostato permanentemente. Il documento richiesto e' stato spostato ad un nuovo indirizzo e gli e' stato assegnato un nuovo URL (alcuni browser puntano automaticamente al nuovo URL)
- Found 302 − Spostato temporaneamente. Il documento richiesto e' stato temporaneamente spostato ad un nuovo indirizzo
- Method 303
- Not Modified 304 − Quando il client effettua una richiesta di GET condizionata ed il documento richiesto non e' stato modificato
- Bad Request 400 − La richiesta non puo' essere soddisfatta dal server in virtu' di un errore di sintassi
- Unauthorized 401 − Errore di sintassi o richiesta impossibile da soddisfare perche' e' stata richiesta l' autenticazione dell' utente
- Payment Required 402 − E' richiesto un pagamento per soddisfare la richiesta
- Forbidden 403 − La richiesta e' proibita (per esempio non hai il corretto permesso per eseguire uno script CGI)
- Not Found 404 − Il documento non e' stato trovato (URL errato o documento spostato in una directory diversa)
- Internal Error 500 − Il server ha incontrato una condizione inaspettata
- Not Implemented 501 − Il server non supporta il servizio richiesto
- Service temporarily overloaded 502 − Il server non puo' soddisfare la richiesta a causa di un sovraccarico del server
- Gateway timeout 503 − Il server tenta di accedere a qualche altro servizio ma non ottiene risposta

<<<indietro Home CGI avanti >>>

Un paio di parole sulla sicurezza degli script CGI

Le form e l' autorizzazione degli script CGI sono probabilmente il piu' importante rischio per un server web. Infatti gli script CGI sono programmi che GIRANO sul tuo server. Il problema e': quando la CGI accetta l' input di un utente, e lo script non controlla le informazioni ricevute, beh, quelle informazioni potrebbero trasformarsi in pericolosi COMANDI, ed il server li eseguira! Percio' attenzione ai CGI liberamente disponibili sulla rete. Prima di installare un nuovo script CGI che non conosci, dovresti controllarlo. Ricorda queste regole:

1. Controlla l' input degli utenti
2. Non installare script CGI troppo voluminosi
3. Evita assolutamente i programmi generati automaticamente
4. Evita le chiamate alla shell Unix

Ma vediamo che tipo di rischi si corrono in seguito di attacchi di utenti malintenzionati:

1. Sendmail e Perl
2. Chiamate alla shell Unix

Sendmail e Perl

Ok, supponi di voler sviluppare un servizio basato su script CGI, con il quale gli utenti possono accedere ad un database ed effettuare delle ricerche per poi ricevere i risultati via email (ad esempio i nuovi libri arrivati nella tua libreria e disponibili per consultazioni). Al fine di raggiungere questo scopo devi costruire questo comando:

```
sendmail −t utente@indirizzo.com  </.nuoviarrivi
```

Il comando qui sopra invia il file 'nuoviarrivi' all' indirizzo email dell' utente specificato (per esempio 'utente@indirizzo.com'). Cosi' costruisci una form con un campo di input dove l' utente puo' specificare il suo indirizzo. Bene, il problema e': se il tuo script non controlla l' input degli utenti, un utente malintenzionato potrebbe compilare la tua form in questo modo:

```
utente@indirizzo.com </etc/passwd;
```

Cosa e' successo? Bene, prima di tutto devi sapere una cosa: il file passwd e' il file piu' ambito ed agognato per gli utenti malintenzionati, infatti contiene tutti gli userid e le password...di tutti gli utenti che hanno un account sull' host dove gira il server...Percio' cosa e' successo? E' semplice: l' utente malintenzionato sta chiedendo al tuo server: 'per favore, invia il file delle password al mio indirizzo...grazie...'. ma come e' possibile? Il tuo script invia solamente della posta agli indirizzi specificati...Certo, ma il tuo script 'legge' l' input dell' utente e lo considera un indirizzo email. L' interprete Perl sa che ';' significa la fine di un comando, cosi capisce che deve eseguire 2 comandi. Il comando originario diventa:

```
sendmail −t utente@indirizzo.com </etc/passwd; </.nuoviarrivi
```

E l' interprete Perl eseguira':

1. sendmail −t utente@indirizzo.com </etc/passwd
2.

```
</.nuoviarrivi
```

Ovviamente il secondo comando produrra' un errore, ma il primo verra' eseguito!

Chiamate alla shell Unix

L' interprete Perl o anche il linguaggio C, hanno delle funzioni che consentono di effettuare delle chiamate di sistema. In altre parole il tuo script potrebbe eseguire qualsiasi comando tramite alcune funzioni particolari, come system(), popen() od eval(). Usa la funzione Perl exec invece di system().

Come posso evitare questi problemi?

Controlla l' input degli utenti. Dovresti ammettere solo cifre e lettere e piazzare un bel '\' prima di qualsiasi carattere 'strano'. Per esempio: il simbolo '<' permette la redirezione dell' output, percio' in un sistema Unix e' un carattere speciale. Comunque se inserisci un '\' prima, diventa il comune simbolo minore (cioe', significa 'minore di'). Evita le funzioni 'eval', 'popen()' e 'system()'. Non permettere le chiamate alla shell Unix. Non installare script CGI troppo voluminosi (specie se sconosciuti. Infatti maggiori sono le dimensioni di un programma maggiore e' il numero di 'bug' contenuti al suo interno). Non installare assolutamente script CGI sconosciuti (controllali prima). Evita assolutamente la generazione automatica di programmi (come puoi controllare qualcosa che non esiste ancora?).

<<<indietro Home CGI avanti >>>

Links a risorse CGI e Perl

- www.perl.com
- www.softlab.ntua.gr/unix/perl_manual
- www.khoros.unm.edu/staff/neilb/perl/vhll/slide01.html
- www.jmarshall.com/easy/cgi
- www.metamorphix.com/cgi.html
- www.best.com/~hed/und/cgi−faq
- www.camtech.com.aw/jemtek/cgi/lib
- www.mispress.com
- www.mep.com
- www.lne.com
- www.hcs.harvard.edu

<<<indietro Home CGI