7. SERVERUL WEB – CONFIGURARE AVANSATA SI SECURIZARE

7.1. Protocolul HTTP

7.1.1. Anatomia mesajelor HTTP

Dupa cum se cunoaste, comunicatia HTTP intre clientul si serverul web consta in schimburi de mesaje - cereri formulate de client si raspunsuri corespunzatoare ale serverului. Vom analiza in continuare compozitia acestora, evidentiind elementele de interes din perspectiva administrarii de server web.

Mesajul HTTP contine, in aceasta ordine:

1. o prima linie care constituie cererea sau raspunsul, in functie de caz
2. (optional) una sau mai multe linii header (antet) – informatii aditionale care pot fi folosite de serverul web sau de aplicatiile sale conexe pentru a lua decizii (vezi mai jos sectiunea despre headere)
3. o linie goala
4. (optional) continut – spre exemplu, sursa HTML a unei pagini web transmisa de serverul web clientului, sau continutul unui formular HTML expediat de catre client serverului. Multe mesaje HTTP (in general cererile) sunt alcatuite doar din prima linie si eventuale headere, fara a dispune si de continut

Nota: *spre deosebire de alte protocoale, ale caror mesaje erau in format binar, mesajele HTTP sunt clear-text – vizualizand un mesaj HTTP capturat din retea cu un viewer/editor de text vom vedea in clar cererea/raspunsul si headerele.*

O cerere HTTP are un format asemanator cu cel din exemplul urmator:

```
GET /index.html HTTP/1.1
Host: www.example.ro
Connection: close
User-Agent: Web-sniffer/1.0.36 (+http://web-sniffer.net/)
Accept-Encoding: gzip
Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7
Cache-Control: no
Accept-Language: de,en;q=0.7,en-us;q=0.3
Referer: http://web-sniffer.net/
```

Prima linie reprezinta asa-numitul “request line” si contine:

- tipul de cerere HTTP (GET)
- URI-ul resursei dorite (calea web catre acea resursa)
- versiunea de protocol HTTP suportata de catre client (HTTP/1.1)

Toate celelalte linii reprezinta headere HTTP. Cererea nu include continut.

Un raspuns HTTP este de forma:

```
HTTP/1.1 200 OK
Date: Wed, 28 Jul 2010 08:22:49 GMT
Server:      Apache/2.2.11 (Unix) mod_ssl/2.2.11 OpenSSL/0.9.8e-fips-rhel5
mod_auth_passthrough/2.1 mod_bwlimited/1.4 FrontPage/5.0.2.2635 PHP/5.2.9
X-Powered-By:PHP/5.2.9
Connection:  close
Transfer-Encoding:  chunked
Content-Type:text/html

<HTML>
    <BODY>
[....restul continutului a fost omis...]
```

Prima linie – asa-numitul “status line” - contine:

- versiunea de protocol HTTP suportata de server (HTTP/1.1)

- un cod de raspuns HTTP, indicand rezultatul operatiei solicitate de client (200)
- un scurt mesaj descriptiv al codului de raspuns (OK)

Liniile ramase sunt, din nou, headere HTTP, urmate de linia goala si de sursa paginii web transmise clientului.

7.1.2. Tipuri de cereri HTTP

Orice cerere HTTP este adresata unei anumite resurse (elementul URI din request line). Tipul de cerere (primul element al liniei) indica serverului operatia pe care trebuie sa o aplice resursei dorite si felul in care trebuie sa raspunda la solicitare. Iata cateva tipuri de cereri:

- **GET** – cea mai des intalnita cerere HTTP. GET este gandit pentru a obtine de la server resursa dorita fara a produce modificari asupra datelor aflate pe acesta; astfel, o aceeasi cerere GET poate fi repetata de oricate ori fara efecte secundare. Cererile de tip GET sunt cele generate atunci cand scriem o adresa in bara browserului sau cand dam click pe un link
- **HEAD –** este echivalenta cu o cerere GET, insa serverul va raspunde doar cu status line-ul si headerele corespunzatoare resursei solicitate de client, fara a trimite si continutul. Este o modalitate de obtinere a headerelor fara a ocupa banda inutil prin transferul intregului continut; clientul poate descoperi astfel, de exemplu, ce tipuri de continut poate serverul primi/oferi (vezi mai jos headere)
- **POST** – acest tip de cerere este gandit pentru a trimite serverului date si a produce modificari asupra informatiilor stocate pe acesta. Cea mai intalnita utilizare a lui POST este expedierea datelor dintr-un formular HTML, caz in care cererea poate determina, spre exemplu, adaugarea sau stergerea de informatii dintr-o baza de date; de aceea repetarea unei cereri POST poate avea efecte nedorite. Din acest motiv browserele avertizeaza atunci cand incercam sa apasam Reload in cazul unei pagini obtinute prin POST
- **PUT** – folosit pentru a modifica (uploada) fisiere pe serverul web, daca acesta o permite
- **DELETE** – folosit pentru a sterge fisiere de pe serverul web, daca acesta o permite

Serverele web permit in general filtrarea in functie de tipul de cerere – putem interzice unele tipuri de cereri, le putem permite selectiv numai pentru anumiti clienti, putem limita numarul de cereri de un anumit tip in unitatea de timp etc.

7.1.3. Coduri de raspuns HTTP

Codul de raspuns HTTP indica clientului rezultatul operatiei solicitate anterior printr-o cerere HTTP. Codurile posibile cad sub incidenta a 5 categorii:

- **1xx** - “Informational”. Indica un raspuns provizoriu din partea serverului, acesta asteptand clientul sa continue o serie de cereri (ex: clientul uploadeaza un fisier mare; el trimite intai serverului cererea POST, comunicandu-i lungimea continutului trimis, iar acesta ii raspunde cu 100 - Continue)
- **2xx** - “Success”. Sunt coduri care indica reusita operatiei solicitate, si care pot diferi in functie de tipul operatiei. Cel mai intalnit este 200 care indica reusita cererilor uzuale GET si POST
- **3xx** - “Redirection” – indica clientului ca trebuie sa efectueze o cerere suplimentara (in general adresata unui alt URI decat cel initial) pentru a obtine resursa dorita. Noul URI este indicat sub forma unui header Location. Iata cateva coduri uzuale:
  - 301 – redirectionare permanenta. Instiinteaza clientul ca resursa dorita nu mai este disponibila la URI-ul cerut, indicandu-i noul URI de la care ea trebuie obtinuta
  - 302 – redirectionare temporara. Resursa dorita de client este momentan indisponibila, acestuia indicandu-i-se noul URI. Clientul va folosi insa pe viitor tot URL-ul vechi pentru a solicita aceasta resursa.

    Nota: *diferenta intre 301 si 302 conteaza mai ales in cazul in care clientul serverului web este un server proxy, care efectueaza solicitari in numele clientilor sai. Interesul proxy-ului este ca, daca un URI nu mai este de actualitate, sa nu il mai solicite de fiecare data in mod inutil; de aceea, in cazul primirii unei redirectionari permanente, proxy-ul retine indisponibilitatea URI-ului vechi si il va folosi in continuare numai pe cel nou, chiar si atunci cand clientii i-l solicita pe cel vechi.*

  - 303 (see other) – resursa dorita nu mai exista la URI-ul curent, insa poate fi gasita la un alt URI, accesibil folosind GET

- 304 (not modified) – resursa ceruta nu a fost modificata. Acest cod de raspuns este folosit pentru a instiinta un client (de obicei un server proxy) ca resursa dorita nu s-a mai modificat de la ultima sa obtinere

- **4xx** - "Client Error". Indica ceea ce serverul presupune a fi o eroare din partea clientului. Coduri uzuale:
  - 401(Unauthorized) – clientul nu este autorizat sa primeasca resursa ceruta. Raspunsul include un header WWW-Authenticate prin care se solicita clientului autentificarea
  - 403 (Forbidden) – accesul clientului la resursa dorita a fost interzis din configurarea serverului (ex: prin intermediul unei directive *Deny from* in cazul Apache)
  - 404 (Not Found) – resursa dorita de client nu a fost gasita
  - 410 (Gone) – resursa dorita de client nu mai este disponibila (aplicabil atunci cand serverul isi poate da seama ca acea resursa a existat candva insa nu mai exista in momentul sosirii cererii)
- **5xx** - "Server Error". Indica o eroare din partea serverului, foarte des una de configurare. Cea mai des intalnita este 500 Internal Server Error, cu semnificatia: cauze neasteptate impiedica serverul sa onoreze cererea primita (spre exemplu, in cazul unui fisier cu useri absent sau invalid in cazul autentificarii HTTP)

7.1.4. Headere HTTP

Headerele HTTP reprezinta meta-informatii incluse in cererea sau raspunsul HTTP, pe care serverul sau clientul le pot folosi pentru a lua decizii. Fiecare header ia forma unei linii plasate in cadrul mesajului HTTP intre request/status-line si linia goala. O linie header este de forma *Nume-Header: valoare*.

Fara a incerca o prezentare exhaustiva a headerelor HTTP, vom trece in revista cateva dintre cele mai importante categorii de headere HTTP:

- headere ce descriu continutul inglobat in mesaj. Exemple sunt *Content-Type* (tipul MIME al continutului), *Content-Length* (lungimea continutului transmis), *Content-Disposition* (modul dorit de manipulare a continutului transmis) etc.
- headere ce descriu capabilitati ale clientului/serverului. Amintim *Accept* (descrie tipurile MIME de continut pe care clientul/serverul sunt dispune sa le primeasca), *Accept-Language* (limbile preferate), *Accept-Encoding* (modalitati de reprezentare a continutului acceptate – ex: continutul poate fi comprimat/decomprimat folosind gzip, deflate) etc.
- headere folosite pentru setarea si transmiterea cookie-urilor: *Set-Cookie, Cookie, Cookie2*
- headere folosite in autentificarea HTTP: *WWW-Authenticate* (prin care serverul solicita clientului autentificarea), *Authorization* (prin care clientul furnizeaza informatiile de autentificare)
- informatii despre client/server: *Server* (contine numele serverului si eventuale detalii despre acesta, in functie de configurare), *User-Agent* (descrie clientul – daca acesta este un browser, aflam din acest header numele si versiunea lui)
- headerul de redirectionare: *Location*, folosit in asociere cu codurile de raspuns 3xx
- headerul *Host*, prin intermediul caruia clientul instiinteaza serverul ce nume DNS a folosit pentru a-l accesa. Acest header este indispensabil functionarii virtual hosting-ului named-based
- headerul *Referer* – indica serverului URL-ul de la care provine clientul (subiectul cererii imediat anterioare a clientului), care poate pointa catre acelasi server sau altul

7.2. Pagini de eroare personalizate

Actiunea implicita a serverului web atunci cand semnalizeaza clientului o eroare este sa ii trimita acestuia o pagina minimalista ce contine un mesaj de eroare prestabilit. In masura in care doreste o prezentare diferita a erorilor, administratorul serverului web poate folosi directiva *ErrorDocument* pentru a alege in ce fel sa fie afisate diferitele tipuri de erori. Personalizarea se poate face pentru fiecare cod in parte folosind urmatoarea sintaxa:

```
ErrorDocument       cod_raspuns_HTTP     document
```

Documentul (al doilea parametru al directivei) poate fi:

- un mesaj specificat de catre administrator chiar in cadrul directivei *ErrorDocument*:

```
ErrorDocument 403   "<h1>Accesul interzis</h1>"
```

- o redirectionare catre un document sau script local, care contine/genereaza mesajul de eroare. **Atentie! Ultimul argument este un URL-path, nu o cale in sistemul de fisiere!**

```
ErrorDocument 404   /erori/404.html
ErrorDocument 401   /erori/generate_error_page.php?code=401
```

- redirectionare catre un URL extern:

```
ErrorDocument 403   http://alt.server.ro/erori/error.php?code=403
```

7.3. Securizare generala a serverului Apache

7.3.1. Limite

Parte a securizarii oricarui server este stabilirea de limite pentru diversele resurse consumate. Identificam urmatoarele aspecte care accepta limitari:

- timeout-uri pentru conexiuni – timpul cat este serverul dispuns sa tina conexiunea deschisa in lipsa vreunei cereri din partea clientului. Orice conexiune inseamna din punct de vedere al serverului resurse consumate, si a pastra conexiunile deschise timp indelungat (in conditiile in care clientii din cealalta parte nu mai au activitate) poate duce la epuizarea in timp a resurselor serverului. Putem seta timeout pentru:
  - conexiuni obisnuite:

```
TimeOut 300  # valoarea este in secunde
```

  - conexiuni keep-alive (cele care sunt “refolosite” de catre clienti prin trimiterea mai multor cereri HTTP in cadrul aceleiasi conexiuni TCP:

```
# activare conexiuni persistente
KeepAlive On

# maximul de cereri per conexiune persistenta
MaxKeepAliveRequests 100

# pauza maxima intre doua cereri consecutive din cadrul unei conexiuni persistente
KeepAliveTimeout 15
```

- limite impuse mesajului HTTP

```
# dimensiunea maxima a mesajului HTTP; in acest caz, nelimitata:
LimitRequestBody 0

# numar maxim de headere prezente intr-o cerere
LimitRequestFields 100

# lungimea maxima a unui header
LimitRequestFieldsize 8190

# lungimea maxima a request line-ului
LimitRequestLine 8190
```

- numar de servere aflate in asteptare. Fiecare nou client Apache este servit de un nou proces/thread (in functie de configurare). Pentru a elimina penalitatea de viteza introdusa de pornirea a mai multe procese/thread-uri atunci cand mai multi clienti se conecteaza intr-un interval scurt de timp, Apache mentine permanent cateva procese/thread-uri deja pornite, in asteptare. Numarul lor este reglabil:

```
# pastram intotdeauna minim 5 procese server pornite, pregatite de a servi clienti
MinSpareServers 5

# ...insa nu depasim 10 astfel de procese
MaxSpareServers 10

# la pornirea Apache vor fi activate 5 servere
StartServers 5
```

- numar de clienti, global sau per sub-proces Apache:

```
# numarul total de clienti conectati simultan
MaxClients 150

# fiecare sub-proces va fi oprit si inlocuit cu unul nou dupa servirea acestui numar de cereri
MaxRequestsPerChild 1000
```

7.3.2. Informatii oferite clientilor

Serverul web poate oferi clientilor tot felul de informatii despre sine – fie ca parte a paginilor web oferite, fie in cadrul headerelor transmise clientului. Aceste informatii pot fi folosite de un eventual atacator pentru a determina softul de server folosit, versiunea acestuia, sistemul de operare aflat la baza etc. ceea ce ii usureaza mult planificarea unui atac.

Putem controla informatia transmisa clientului in urmatoarele aspecte:

- serverul poate include o semnatura in unele pagini auto-generate (ex: listing-uri de directoare, pagini de eroare etc). Semnatura contine numele serverului si adresa de email a administratorului. Putem modifica adresa de email sau putem suprima complet semnatura:

```
# adresa de email care apare in unele pagini auto-generate de Apache
ServerAdmin admin@server.ro

# suprimarea semnaturii:
ServerSignature Off
```

- serverul transmite clientului informatii despre sine prin intermediul headerului *Server*. Putem alege sa transmitem doar numele softului de server, sau sa transmitem o valoare arbitrara:

```
# headerul va contine doar numele produsului (Server: Apache)
ServerTokens ProductOnly
# valori alternative: Major, Minor, Minimal, OS, Full

# headerul Server va contine o valoare aleasa de noi, cu conditia sa fie incarcat mod_header
Header set Server "Microsoft-IIS/5.0"
```

7.3.3. Securizarea accesului la fisierele din afara document root-ului

Ca si in cazul firewall-urilor, la configurarea serverului web este sanatos sa fie implicit interzis accesul la toate resursele si apoi sa fie permis in mod controlat numai pentru cele dorite. Interdictia se aplica directorului / (si va fi automat valabila in toate subdirectoarele); ea va fi urmata de eventuale relaxari pentru directoare punctuale, in functie de necesitati:

```
# interzicere acces in tot sistemul de fisiere
<Directory />
    Order Allow,Deny
    Deny from all
    Options None
</Directory>

# relaxare politici acces pentru DocumentRoot
```

```
<Directory /var/www>
      Order Allow,Deny
      Allow from all
</Directory>
```

7.3.4. Securizare acces la fisierele de configurare per-director

Una dintre facilitatile Apache (descrisa pe larg intr-un capitol ulterior al acestui material) este folosirea de fisiere de configurare per-director. Numele implicit al acestora este *.htccess* (dar el poate fi modificat din httpd.conf!). Nu dorim ca aceste fisiere de configurare sa fie livrate din greseala clientilor, motiv pentru care interzicem explicit accesul la ele:

```
<FilesMatch ^\.ht>
      Order Allow,Deny
      Deny from all
</FilesMatch>
```

7.3.5. Dezactivarea optiunilor nefolositoare sau periculoase

Directiva Options controleaza optiunile disponibile per-director. Forma sa generala este urmatoarea:

```
Options +optiune1 +optiune2 -optiune3 -optiune4 .....
```

unde – dezactiveaza optiuni si + le activeaza. Dintre optiunile posibile ne vom concentra asupra catorva de interes:

- **FollowSymlinks** – permite serverului sa urmeze symbolic link-urile. Este o optiune potential periculoasa, deoarece un symlink poate trimite in afara DocumentRoot-ului; preferabila este utilizarea unui Alias cu control strict atunci cand dorim ca clientul sa aiba acces in afara DocumentRoot
- **FollowSymLinksIfOwnerMatch** – daca totusi este necesara urmarea symlink-urilor, putem impune ca ele sa fie urmate de catre server numai daca owner-ul symlink-ului coincide cu cel al fisierului tinta
- **Indexes** – activeaza generarea automata a listing-ului atunci cand clientul solicita un director. Optiunea poate oferi clientului informatii prea bogate si de aceea trebuie activata numai in caz de nevoie
- **ExecCGI** – permite executia scripturilor CGI in interiorul directorului. Optiunea trebuie activata numai in caz de necesitate
- **All** – desemneaza toate optiunile in afara de MultiViews (folosita in negocierea de continut)

```
# dezactivarea tuturor optiunilor
<Directory />
      Options -All -MultiViews
      # alternativ: Options None
</Directory>
```

7.4. Configurarea logging-ului in Apache

7.4.1. Tipuri de loguri

Serverul Apache poate genera urmatoarele tipuri de loguri:

- loguri de baza
  - **error log** – este logul in care se consemneaza erorile/avertizarile legate de pornirea si functionarea serverului. Exista un singur astfel de log global; in plus, atunci cand se foloseste virtual hosting, poate exista un singur log de erori per host virtual. Formatul logului de erori este prestabilit si nu poate fi modificat de catre administrator, insa acesta din urma poate alege gradul de detaliu al informatiilor cuprinse in log
  - **access log** (numit ocazional si transfer log) – este logul in care se consemneaza rezultatul cererilor primite de la clienti. Inregistrarea corespunzatoare unei cereri este consemnata *dupa* servirea cererii si poate include orice fel de detalii despre cerere si rezultatul acesteia (inclusiv cod de raspuns HTTP, mesaj etc). Formatul

inregistrarilor poate fi ales de catre administrator. Pot exista mai multe astfel de fisiere log per context (main, VH) si ele vor fi folosite simultan

- loguri aditionale, specifice anumitor module. Cateva exemple:
  - **rewrite log** – consemneaza informatii legate de transformarile aplicate URL-urilor pe care le efectueaza modulul mod_rewrite
  - **forensic log** – logul generat de modulul *mod_log_forensic*, care pentru fiecare cerere produce doua inregistrari – una imediat dupa primirea cererii si alta dupa servirea acesteia
  - **SSL log** – consemneaza informatii legate de functionarea modulului mod_ssl in versiunile mai vechi de Apache. In versiunile noi, informatiile legate de SSL pot fi consemnate separat folosind o directiva aditionala CustomLog

7.4.2. Logul de erori

Logul de erori are format prestabilit; administratorul nu poate modifica decat doua aspecte legate de consemnarea erorilor:

- destinatia informatiilor de logging de eroare, care poate fi:
  - un fisier
  - syslog
  - un executabil care prelucreaza informatiile
- nivelul de importanta minim pe care trebuie sa il aiba o informatie pentru a fi consemnata in loguri (echivalent cu *severity* de la serverul DNS)

Directivele care configureaza aceste aspecte sunt cele din exemplele de mai jos:

```
# nivelul de importanta minim consemnat
LogLevel debug
# posibilitati: debug, info, notice, warn, error, crit, alert, emerg

# trimiterea erorilor catre syslog, cu specificarea facility-ului dorit
ErrorLog syslog:local7

# scrierea erorilor intr-un fisier gestionat de Apache
ErrorLog /var/log/apache/error.log
# alternativa nerecomandata - suprimare erori: ErrorLog /dev/null

# trimiterea erorilor catre un script/executabil care le va prelucra
ErrorLog "|/usr/local/sbin/prel-logs"
```

ErrorLog-ul este unul singur pentru contextul global al serverului (“radacina” fisierului de configurare). In cazul virtual hosting poate fi definit un sigur error log pentru fiecare server virtual. Daca avem totusi nevoie sa logam erori catre mai multe destinatii simultan, va trebui sa rezolvam problema prin mijloace externe, trimitand logurile catre un executabil, conform ultimului dintre exemplele de mai sus.

Nota: *putem emula optiunea -g de la serverul DNS (rulare in mod debug si in foreground) trimitand ErrorLog-ul catre terminalul curent (al carui fisier corespondent poate fi aflat cu comanda* tty*) si ruland serverul cu optiunea -X.*

7.4.3. Logul de acces

Access log-ul este cel care consemneaza rezultatele tratarii cererilor primite de la clienti, dupa servirea acestora. El este disponibil numai daca a fost incarcat modulul *mod_log_config* . Access log-ul este configurabil, atat ca informatii consemnate cat si ca destinatie a informatiilor de logging. Putem configura mai multe destinatii de logging per context (cel global sau in interiorul VH-urilor) si ele vor actiona independent – serverul va scrie informatii in toate.

Modulul *mod_log_config* introduce urmatoarele directive:

- **LogFormat** – este folosita pentru a defini compozitia inregistrarilor consemnate in loguri. Folosind de mai multe ori aceasta directiva putem defini mai multe formate, fiecare purtand un nume prin intermediul caruia va fi referit in alte directive din fisierul de configurare (vezi mai jos)

- **CustomLog** – specifica o destinatie de logging si formatul inregistrarilor din acea locatie

Directiva *LogFormat* are sintaxa urmatoare:

```
LogFormat "specificator_format" nume_format
```

Specificatorul de format este un sir de caractere ce contine secvente de forma %string; lista de secvente posibile este prezentata in anexa 1 a materialului.

Directiva *CustomLog* are forma:

```
CustomLog    destinatie    format|nume_format        <env=<!>variabila>
```

Destinatia poate fi un nume de fisier sau un pipe (ca in cazul lui *ErrorLog*). Al doilea parametru poate fi fie un string ce defineste formatul (aceeasi sintaxa ca in cazul lui LogFormat), fie numele unui format definit anterior cu *LogFormat*. Ultimul argument, optional, permite conditionarea consemnarii cererii in log de existenta unei variabile de mediu Apache (detalii in sectiunea despre variabile).

Iata cateva exemple:

```
# FORMATE CELEBRE

# CLF (common Log Format)
LogFormat "%h %l %u %t \"%r\" %>s %b" common

# combined log format
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" combined

# FORMATE CUSTOM

# crearea unui log separat cu browserele folosite de clienti, pe baza headerului User-Agent
LogFormat "%{User-Agent}i"       browser
CustomLog    /var/log/apache/browsers.log     browser
# alternativ: CustomLog    /var/log/apache/browsers.log     "%{User-Agent}i"

# log formatat care consemneaza una sub alta request line, moment de timp si status:
LogFormat "Data/ora: %t \n\tRequest: %r \n\t Status: %s \n"       ascii_art
CustomLog    /var/log/apache/nice.log   ascii_art
```

Nota: *exista si directiva TransferLog, care este identica cu CustomLog ca format si utilitate, cu exceptia faptului ca nu accepta ca format de log decat numele unui format definit anterior.*

7.5. Securizarea comunicatiilor serverului folosind SSL

7.5.1. Cerinte

Pentru a putea folosi SSL/TLS in scopul criptarii/autentificarii comunicatiilor serverului este necesara intai de toate includerea in server a modulului **mod_ssl**. La instalarea serverului din pachete, modulul este prezent de obicei sub forma unui pachet separat ce contine fisierul mod_ssl.so si eventuale configurari legate de acesta; la compilarea din surse este necesara includerea explicita a modulului folosind optiunea corespunzatoare a scriptului *configure*:

```
./configure -prefix=/usr/local/apache --enable-so --enable-ssl ......alte optiuni.....
```

Pentru a putea comunica securizat, serverul va avea nevoie de o pereche de chei (privata/publica) si de un certificat digital care sa ateste corespondenta intre cheia publica si identitatea serverului. Certificatul este necesar pentru ca serverul sa se autentifice fata de clienti, acestia intrand astfel si in posesia cheii publice a serverului. Cheia privata este necesara pentru ca serverul sa poata decripta datele trimise de clienti (criptate cu cheia sa publica).

7.5.2. Configurare

Odata serverul instalat si cheile/certificatul pregatite, configurarea presupune urmatorii pasi:

- schimbarea portului pe care asculta serverul sau activarea unui port suplimentar. Clientii care acceseaza serverul pe portul 80 se asteapta ca acesta sa comunice necriptat, de aceea o minima masura este sa mutam serverul pe portul 443 (portul default de HTTPS). In cazul in care serverul va oferi atat serviciu criptat cat si necriptat, vom folosi doua directive Listen pentru a deschide ambele porturi (80 si 443)
- activarea motorului de SSL – se realizeaza cu directiva *SSLEngine*, care are argumente posibile *On* sau *Off*. Activarea se poate realiza la nivel de intreg server - plasand directiva in contextul global - sau per virtual host. Nu se poate face activarea SSL per director, location etc
- configurarea cailor catre cheia privata si certificat, folosind directivele *SSLCertificateFile* si *SSLCertificateKeyFile*
- (optional, daca certificatul este semnat de un intermediate CA) configurarea caii catre fisierul care contine certificatele tuturor CA-urilor intermediare, folosind directiva *SSLCertificateChainFile*. Aceasta lista de certificate va fi transmisa clientului ca parte a handshake-ului SSL/TLS, pentru ca acesta sa dispuna de toate informatiile necesare autentificarii serverului

Exemplu:

```
# deschidere port HTTPS
Listen 443

# activare SSL
SSLEngine On

# locatia fisierului ce contine certificatul
SSLCertificateFile /etc/apache/apache.crt

# locatia fisierului ce contine cheia privata
SSLCertificateKeyFile /etc/apache/apache.key

# (daca e cazul) locatia fisierului ce contine certificatele CA-urilor intermediare, concatenate
SSLCertificateChainFile /etc/apache/CAbundle.pem
```

Daca serverul ofera atat serviciu criptat cat si necriptat, nu putem activa global motorul de SSL, ci este necesara crearea unui virtual host care actioneaza pe portul 443:

```
Listen 80
Listen 443
[...lista de directive globale...]
<VirtualHost *:443>
      SSLEngine     On
      [...lista de directive SSL necesare (cheie,certificat etc)...]
      [...lista celorlalte directive...]
</VirtualHost>
```

Nota: *in cazul unui scenariu de tip "doua servere intr-unul" ca cel de mai sus, putem efectua pornirea conditionata a serverului virtual SSL incluzand intregul VH intr-o directiva <IfDefine> si pasand parametrul corespunzator la pornirea serverului in functie de necesitati.*

7.5.3. Optiuni SSL

Modulul mod_ssl introduce o directiva asemanatoare directivei Options, cu acelasi mod de utilizare: SSLOptions. Aceasta primeste unul sau mai multe argumente (optiuni individuale) ce pot fi precededate de simbolurile + sau - pentru activare respectiv dezactivare. Optiuni de interes:

- **FakeBasicAuth** – emuleaza autentificarea de tip Basic, transformand DN-ul din certificatele de client in username (vezi sectiunea despre controlul accesului folosind SSL)

- **StrictRequire** – atunci cand intr-un director sunt folosite mai multe mecanisme de autorizare (ex: atat in functie de IP client cat si de parametri SSL), daca modulul SSL interzice accesul atunci el are ultimul cuvant (decizia sa nu poate fi anulata de alt modul de autorizare, asa cum se intampla in cazul folosirii lui Satisfy any)

7.5.4. Securizare SSL

A securiza Apache din punct de vedere SSL presupune cel putin doua aspecte:

- controlul listei de protocoale permise. Se realizeaza cu directiva SSLProtocol urmata de unul sau mai multe argumente, care pot fi SSLv2, SSLv3, TLSv1. Fiecare argument poate fi precedat de simbolul + pentru activare si – pentru eliminare. Exista suplimentar argumentul posibil *All* care este echivalent cu "+SSLv2 +SSLv3 +TLSv1":

```
# protocoale permise: doar SSLv3 si TLS
SSLProtocol All -SSLv2

# protocol permis: doar TLS
SSLProtocol TLSv1
```

- controlul listei de algoritmi criptografici acceptati in comunicatiile cu clientii. Se realizeaza cu directiva SSLCipherSuite urmata de unul sau mai multe argumente ce definesc cifruri sau clase de cifruri. Lista completa de cifruri poate fi gasita la http://httpd.apache.org/docs/2.2/mod/mod_ssl.html#sslciphersuite. Fiecare cifru poate fi precedat de:
  - \+ adauga cifrul in lista de cifruri admise
  - \- elimina cifrul, dar cu posibilitatea ca el sa fie adaugat din nou mai tarziu in cadrul listei
  - ! elimina cifrul definitiv, fara posibilitatea adaugarii ulterioare

```
# toate cifrurile disponibile, mai putin cele ce securitate scazuta si cele fara criptare
SSLCipherSuite ALL:!LOW:!NULL

# doar cifrurile cu securitate ridicata si cele RSA
SSLCipherSuite HIGH:RSA
```

Nota: *setul de cifruri ales poate fi verificat cu comanda openssl cipher, care expandeaza eventualele alias-uri:*

```
# openssl ciphers -v HIGH:MEDIUM:\!SSLv2

EDH-RSA-DES-CBC3-SHA SSLv3 Kx=DH  Au=RSA Enc=3DES(168) Mac=SHA1
EDH-DSS-DES-CBC3-SHA SSLv3 Kx=DH  Au=DSS Enc=3DES(168) Mac=SHA1
DES-CBC3-SHA         SSLv3 Kx=RSA Au=RSA Enc=3DES(168) Mac=SHA1
RC4-SHA              SSLv3 Kx=RSA Au=RSA Enc=RC4(128)  Mac=SHA1
RC4-MD5              SSLv3 Kx=RSA Au=RSA Enc=RC4(128)  Mac=MD5
```

7.5.5. Modalitati suplimentare de control al accesului introduse de mod_ssl

Accesul la resurse poate fi controlat in functie de parametri SSL, in urmatoarele moduri:

- putem impune acces doar prin SSL la anumite resurse, folosind directiva **SSLRequireSSL** in context <Directory>, <Location> etc. Efectul este intoarcerea unui cod 403 (forbidden) pentru clientii care incearca sa acceseze resursele printr-o conexiune necriptata.
- putem stabili seturi de reguli de acces care sa tina cont de diferiti parametri SSL ai sesiunii clientului sau de variabile definite de *mod_ssl* sau de alte module (ex: informatii cuprinse in certificatul serverului sau al clientului, caracteristici ale sesiunii, variabile de mediu Apache etc). Folosim in acest scop directiva **SSLRequire** urmata de o expresie compusa din conditii elementare
- putem impune clientilor sa se autentifice folosind certificate digitale, in doua variante:
  - permitem accesul oricarui client care prezinta un certificat valid
  - permitem accesul anumitor clienti autentificati cu certificate digitale

Vom prezenta in continuare scenariul autentificarii clientilor folosind certificate digitale si al autorizarii lor pe baza identitatii cuprinse in certificat.

7.5.6. Autentificarea si autorizarea clientilor pe baza de certificate digitale

Pentru a autentifica clientii folosind certificate digitale sunt necesare urmatoarele conditii:

- sa obligam clientii sa furnizeze certificat digital in cadrul handshake-ului SSL/TLS. Folosim in ascest scop directiva *SSLVerifyClient require*.
- sa ii furnizam serverului o lista de certificate de CA trusted, astfel incat el sa poata valida certificatele primite de la clienti. Folosim pentru aceasta directiva *SSLCACertificateFile*; fisierul catre care ea pointeaza trebuie sa contina, concatenate, toate certificatele de root CA ce se doresc a fi trusted.

```
SSLVerifyClient require
SSLCACertificateFile /etc/apache/trustedCAbundle.pem
# fisier obtinut prin concatenarea tuturor certificatelor de CA
```

Pana in acest punct, clientii care nu prezinta un certificat valid nu vor putea deschide o conexiune cu serverul. Odata conexiunea deschisa insa, ei au acelasi regim de acces ca orice alt client. Daca in cazul unei conexiuni necriptate puteam autoriza un client in functie de caracteristicile sale (mod_authz_host) sau de username-ul furnizat la autentificare HTTP (mod_authn_*), in cazul unei conexiuni criptate putem folosi si identitatea cuprinsa in certificatul clientului pentru a-i autoriza actiunile.

Filtrarea clientilor in functie de informatiile cuprinse in certificatul prezentat se poate realiza in cel putin 3 moduri:

- folosind directiva *SSLRequire* urmata de o expresie care descrie conditiile ce trebuie indeplinite de clienti pentru a putea capata acces la o resursa
- folosind modulul *mod_rewrite* pentru a seta variabile de control pe baza variabilelor definite automat de *mod_ssl* si apoi folosirea variabilelor de control in *Allow from env/Deny from env*

  Nota: *mod_ssl defineste automat o intreaga serie de variabile de mediu care pot fi folosite pentru controlul accesului, al logging-ului sau pentru reguli de rewrite. Lista lor completa se gaseste in Anexa 2. Valorile acestor variabile pot fi incluse in loguri specificand o secventa de tipul* %{variabila}x *in cadrul LogFormat-ului.*

- folosind optiunea SSL *FakeBasicAuth*, care simuleaza o autentificare HTTP de tip basic. Modulul *mod_ssl* ii va transmite lui *mod_auth_basic* un username egal cu DN-ul clientului asa cum apare el in certificat, impreuna cu parola *password,* iar *mod_auth_basic* va autentifica clientul fara a fi constient de provenienta datelor de autentificare

Detaliem in continuare cea de-a treia solutie, deoarece este cea mai eleganta si scalabila.

In directorul in care se doreste autentificare pe baza de certificat se activeaza optiunea SSL *FakeBasicAuth*, dupa care se configureaza autentificarea HTTP in modul obisnuit:

```
<Directory /var/www/private>
      SSLRequireSSL                  # Daca nu a fost deja setat intr-un context mai larg
      SSLVerifyClient     require    # Idem
      SSLOptions +FakeBasicAuth +StrictRequire
      AuthName "Zona privata - accesul neautentificat este strict interzis"
      AuthType Basic
      AuthUserFile /etc/apache/userdb
      Require valid-user
</Directory>
```

Fisierul */etc/apache/userdb* va fi creat ca de obicei, folosind utilitarul *htpasswd*, si va contine conturile clientilor sub urmatoarea forma:

- username-ul fiecarui client este DN-ul clientului, asa cum apare el in cadrul certificatului trimis catre server (vezi exemplul de mai jos)
- parola este *password*, indiferent de client. In fisier este memorat hash-ul acesteia

Iata un exemplu de fisier:

```
/C=RO/ST=SAI/L=Bucharest/O=InfoAcademy/CN=CLIENT1:E0UIgTzHr5s7o
/C=RO/ST=SAI/L=Bucharest/O=InfoAcademy/CN=CLIENT2:E0UIgTzHr5s7o
```

7.5.7. SSL si name-based virtual hosting

Virtual hosting-ul bazat pe nume in combinatie cu SSL/TLS ridica niste probleme de principiu. Serverul decide din ce VH sa serveasca cererea analizand headerul Host primit de la client (care este obligatoriu incepand cu HTTP/1.1). Pentru a crea o conexiune securizata cu clientul, serverul trebuie sa-i prezinte acestuia un certificat valid, al carui nume sa corespunda cu numele solicitat de client. Aceasta ar presupune sa avem mai multe certificate – cate unul pentru fiecare nume – si sa le configuram in cadrul fiecarui VH. Problema este insa ca serverul afla din ce VH serveste cererea abia dupa ce conexiunea a fost stabilita deja si se primeste cererea HTTP; in momentul stabilirii conexiunii SSL/TLS serverul nu ar sti ce certificat sa aleaga deoarece inca nu a primit de la client headerele HTTP.

Pentru a implementa totusi VH in combinatie cu SSL/TLS exista doua solutii – niciuna dintre ele perfect satisfacatoare:

- folosirea de VH IP-based – ceea ce are dezavantajul consumarii cate unei adrese pentru fiecare VH
- folosirea de certificate cu nume de tip wildcard (valide pentru toate numele dintr-un intreg domeniu) – cu dezavantajul ca 1) un astfel de certificat emis de catre un CA oficial costa mult mai mult decat unul normal si 2) nu toate numele VH-urilor sunt neaparat din acelasi domeniu (vezi cazul companiilor de hosting)

7.6. Variabile de mediu Apache

7.6.1. Principii

Variabilele de mediu Apache sunt variabile interne ale serverului, care pot fi setate conditionat sau neconditionat, global sau per-cerere si de care serverul poate tine cont la tratarea unei cereri. In acest scop exista modulele *mod_env* si *mod_setenvif*.

Variabilele de mediu Apache pot proveni din:

- environment-ul shell-ului din care a pornit Apache
- setate manual, neconditionat, din fisierul de configurare
- setate conditionat de catre alte module ale serverului (mod_setenvif, mod_rewrite, mod_ssl etc.)
- variabile speciale, care afecteaza felul in care serverul trateaza anumiti clienti (ex: nokeepalive, force-response-1.0 etc.)

7.6.2. Modalitati de definire si stergere a variabilelor

Pentru a defini variabile de mediu, administratorul are la dispozitie cel putin doua module Apache – *mod_env* si *mod_setenvif* – impreuna cu directivele introduse de acestea:

- mod_env – directive:
  - **SetEnv** – setare neconditionata a unei variabile de mediu (ATENTIE! SetEnv actioneaza tarziu, dupa RewriteCond si SetEnvIf!)
  - **PassEnv** – preluarea valorii unei variabile de mediu din sistemul de operare (si pasarea mai departe catre aplicatiile CGI, daca este cazul)
  - **UnsetEnv** – desfiintare variabila anterior definita
- mod_setenvif – directive:
  - **SetEnvIf** – setare conditionata a unei variabile, numai daca atributul testat corespunde cu regex-ul specificat
  - **SetEnvIfNoCase** – analog cu SetEnvIf, insa compararea se face case insensitive
  - **BrowserMatch –** echivalent cu a aplica directiva SetEnvIf valorii headerului User-Agent
  - **BrowserMatchNoCase** – idem, pentru SetEnvIfNoCase

7.6.3. Definirea conditionata a variabilelor: mod_setenvif

Forma generala a directivei *SetEnvIf* este urmatoarea:

```
SetEnvIf      expresie      regex         [!]variabila<=valoare>
```

Efectul va fi setarea sau resetarea variabilei cerute numai daca expresia se potriveste regex-ului specificat. Daca numele variabilei este precedat de ! atunci ea va fi stearsa.

Expresia (primul argument al lui SetEnvIf) poate fi:

- numele unui header HTTP primit de la client. In aceste conditii, valoarea headerului respectiv va fi cea confruntata cu regex-ul:

```
SetEnvIf      Referer       ^www\.myserver\.ro$        serverul_meu=true
```

- una dintre urmatoarele caracteristici
  - ale serverului:
    - Server_Addr – adresa serverului pe care a fost primita cererea
  - ale clientului:
    - Remote_Host – numele DNS (daca exista) al clientului, obtinut prin rezolutie DNS inversa
    - Remote_Addr – adresa IP a clientului
  - ale cererii
    - Request_URI – URI resursei cerute, asa cum apare el in request line din mesajul HTTP
    - Request_Method – tipul de cerere HTTP
    - Request_Protocol – numele si versiunea protocolului folosit pentru formularea cererii (ex: HTTP/1.1)

```
SetEnvIf      Remote_Addr          ^127\.0\.0\.1$              from_localhost=1
```

- o variabila de mediu definita anterior cu SetEnvIf (anterior in acelasi context, sau intr-un context mai inalt – ex: un director parinte). Se va folosi simplul nume al variabilei pe post de prim parametru al lui *SetEnvIf*

Cel de-al doilea argument al lui SetEnvIf (cel care specifica variabila setata) poate lua urmatoarele forme:

- **nume_variabila** – variabila va fi setata si va primi valoarea 1
- **nume_variabila=valoare** – variabila primeste valoarea ceruta. In cadrul valorii pot fi folosite secventele $1...$9 pentru a referi valorile subexpresiilor din cadrul regex-ului
- **!nume_variabila** – stergerea unei variabile definite anterior

```
# daca numele documentului cerut se termina cu .doc, .txt sau .odt, se va seta o variabila de
# mediu cu valoarea doc, txt sau odt, dupa caz
SetEnvIf      Request_URI    \.(txt|doc|odt)$    tip_document=$1
```

7.6.4. Modalitati de utilizare a variabilelor setate

Odata o variabila setata, ea poate fi utilizata in multiple moduri – iata-le pe cele mai des intalnite:

- directivele Allow/Deny pot pune conditia existentei unei variabile de mediu pentru a controla accesul:

```
# control acces in functie de variabile definite anterior
Allow from env=serverul_meu
Deny from env=robot

# permitere acces numai pentru clientii care folosesc ca browser Opera
SetEnvIfNoCase User-Agent ^.*opera.*$   opera
Allow from opera
```

- logging-ul poate fi facut conditionat, in functie de existenta unei variabile de mediu:

```
CustomLog /var/log/apache/access.log env=!serverul_meu
```

- in mod_rewrite, variabilele de mediu pot fi folosite pe post de conditie de rescriere a unui URL (vezi capitolul dedicat lui mod_rewrite)

7.7. Fisiere de configurare per-director si particularitati

Apache permite administratorului sa foloseasca, in afara fisierului de configurare principal, fisiere de configurare per-director. Directivele dintr-un astfel de fisier se aplica numai directorului in cauza si tuturor subdirectoarelor sale. Fisierul per-director este citit de catre serverul Apache pentru fiecare cerere servita din acel director (nu doar la pornirea serverului, asa cum se intampla cu fisierul de configurare principal); pentru cereri servite din alt director, fisierul nu este nici macar accesat.

Aceasta facilitate are avantajul de putea delega configurarea per-director catre posibili alti utilizatori, non-root. Sa consideram, de exemplu, scenariul unei firme care ofera gazduire de site-uri web, si unul dintre serverele web ale acestei firme: clientii firmei isi upload-eaza site-urile pe server, fiecare in directorul sau, si pot avea nevoie de setari Apache personalizate, aplicabile numai site-ului propriu. Daca nu ar exista fisier de configurare per-director, toate modificarile ar trebui operate in fisierul principal, cu dezavantajul ca 1) presupun manopera din partea administratorului serverului si 2) necesita restart de server pentru a fi introduse in vigoare. Cu fisiere de configurare per-director, clientii isi pot gestiona singuri setarile – desigur, in limitele permise de administratorul serverului.

Dezavantajul fisierelor de configurare per-director este ca sunt ineficiente: ele sunt citite la fiecare livrare a unei resurse din directorul respectiv, spre deosebire de fisierul principal care este citit o singura data, la pornirea serverului. In plus, pentru a determina combinatia de directive ce actioneaza intr-un director, serverul poate fi nevoit sa citeasca fisiere de configurare per-director aflate in directoare parinte, daca ele exista.

Numele implicit al fisierelor de configurare per-director este *.htaccess*; el poate fi modificat folosind directiva AccessFileName:

```
AccessFileName .apacheconf
```

Atentie! *Fisierele .htaccess, aflandu-se sub document root, sunt publice, existand astfel riscul ca ele sa poata fi solicitate si obtinute de catre clienti. De aceea administratorul serverului trebuie sa ia masurile necesare pentru a proteja aceste fisiere, folosind directive de control al accesului.*

Fisierul per-director poate contine majoritatea directivelor disponibile in fisierul principal (cu unele exceptii); in aceste conditii pot aparea suprapuneri – directive prezente atat in fisierul principal cat si in cel per-director, cu efect diferit. Administratorul poate controla in ce masura directivele per-director au intaietate fata de cele din fisierul principal folosind directiva AllowOverride in fisierul principal (vezi mai jos).

Nota: *a plasa directive de configurare intr-un fisier .htaccess are acelasi efect de baza cu a le plasa in cadrul fisierului principal, intr-un container <Directory>! Atunci cand avem de ales, ultima varianta este de preferat, deoarece este mai rapida.*

Pentru a dicta ce directive pot fi folosite in fisierele de configurare per-director (“suprascriind” setarile din fisierul principal), administratorul foloseste in fisierul principal directiva **AllowOverride** urmata de una sau mai multe valori. Valorile pot fi:

- **AuthConfig** – permite folosirea directivelor de autentificare/autorizare (Auth*, Require etc)
- **FileInfo** – permite folosirea directivelor legate de tipul documentelor, setari variabile, headere, cookies, action-uri, rewrite-uri
- **Indexes** – permite folosirea directivelor care controleaza indexul autogenerat al unui director

- **Limit** – permite folosirea directivelor care gestioneaza accesul (Allow, Deny, Order)
- **Options**<=opt1,opt2,...> - permite folosirea directivei Options, cu specificarea setului de optiuni permise
- **None** – valoare folosita pentru interzicerea rapida a tuturor directivelor in fisiere .htaccess

Pagina de documentatie oficiala a fiecarei directive Apache ne ofera, chiar la inceput, o caseta cu informatii care contine, printe altele, doua elemente importante:

- context: aflam astfel daca directiva poate fi utilizata in contextul global (radacina fisierului principal de configurare), in interiorul unui host virtual sau in interiorul unui fisier de configurare per-director
- override: ce optiune AllowOverride este necesara pentru a putea folosi directiva respectiva in fisiere de configurare per-director

Exemplu: pentru a activa, din cadrul unui fisier .htaccess, listing-urile autogenerate intr-un director accesibil numai cu autentificare si pentru a putea customiza aspectul acestor listing-uri, fisierul de configurare principal trebuie sa contina:

```
# httpd.conf
AllowOverride Indexes,AuthConfig,Options=Indexes
```

iar fisierul *.htaccess* va contine:

```
Options +Indexes
# ...plus eventuale optiuni care customizeaza aspectul indexului autogenerat...
AuthType Basic
AuthName "Zona restrictionata"
AuthUserFile /etc/apache/useri.db
Require valid-user
# Remarcam absenta directivei <Directory> in care ar fi fost plasate aceste directive daca
# s-ar fi aflat in fisierul principal
```

7.8. Tratarea cererilor pe baza de criterii complexe (manipulare URL)

7.8.1. Principii. Mecanisme disponibile

Manipularea URL-urilor presupune folosirea de conditii complexe pentru a determina actiunea aplicata cererii primite de la client. In cazul cel mai simplu (cel implicit), la primirea unei cereri, serverul va prefixa URI-ul cuprins in cerere cu valoarea lui DocumentRoot, rezultand calea catre fisierul dorit de client. Administratorul poate modifica acest mod de operare astfel incat el sa rezulte in orice alta actiune posibila.

Pentru a determina actiunea aplicata unei cereri, serverul poate tine cont de aspecte precum:

- URI-ul cerut de client
- tipul de conexiune (HTTP/HTTPS)
- continutul diverselor headere HTTP primite de la client
- caracteristici ale clientului sau conexiunii acestuia cu serverul (adrese/porturi etc)
- variabile de mediu setate in cadrul serverului

Ca actiuni aplicabile unei cereri, serverul poate efectua:

- livrarea unui fisier de pe hard-disk (folosind expresii complexe, ce tin cont de parametrii de tipul celor enumerati mai sus, pentru a-i determina locatia)
- redirectionarea catre o alta resursa, aflata pe acelasi server sau pe un altul
- interzicerea accesului la resursa in cauza

Iata cateva exemple care presupun prelucrare avansata a cererii:

- publicarea de resurse aflate in afara document root-ului
- atunci cand link-urile unui site se schimba, redirectionarea clientilor care cer link-urile vechi catre link-urile noi corespunzatoare
- controlul accesului sau redirectionarea in functie de data/ora

- directionarea clientului catre un site alternativ atunci cand clientul se conecteaza de pe un dispozitiv mobil(telefon, PDA etc)

Mecanismele disponibile administratorului Apache se incadreaza in 3 categorii:

- folosirea de alias-uri (oferite de modulul mod_alias) – realizeaza transformarea unui URI cerut de client intr-o locatie de pe HDD alta decat cea default, fara ca clientul sa fie constient de acest lucru
- folosirea de redirectionari (introduse tot de mod_alias) – trimiterea explicita a clientului catre alt URL/URI. Ca urmare a redirectionarii, clientul va efectua o noua cerere catre resursa indicata
- folosirea modulului mod_rewrite – presupune determinarea actiunii aplicate cererii pe baza de conditii complexe. Include posibilitatile oferite de alias-uri si redirectionari dar permite o flexibilitate net superioara, cu dezavantajul resurselor consumate

Discutam pe rand cele 3 mecanisme.

7.8.2. Alias-uri

Alias-urile sunt deja familiare cititorului din primele module ale cursului. Un alias reprezinta o corespondenta definita manual intre un prefix de URI si o cale in sistemul de fisiere; cu alte cuvinte, in loc sa lasam serverul sa prefixeze URI-ul cererii cu valoarea lui DocumentRoot, ii indicam un alt loc din care sa preia continutul livrat clientului. In luarea deciziei, serverul tine cont exclusiv de URI:

```
Alias    /images     /var/www/pozesite
```

Orice URI care incepe cu */images* va fi livrat din directorul */var/www/pozesite*; spre exemplu, daca clientul solicita */images/gif/1.gif*, atunci serverul va inlocui in URI sirul */images* cu corespondentul */var/www/pozesite* si va livra clientului continutul fisierului */var/www/pozesite/gif/1.gif*.

Exista si o varianta bazata pe regex-uri a directivei Alias, numita AliasMatch, care ne permite sa specificam alias-urile in functie de FORMATUL string-ului ce constituie URI-ul. Primul argument al directivei este un regex, iar ultimul argument poate folosi back references – referinte la valorile sub-expresiilor continute in primul argument:

```
AliasMatch    ^/images/(.*)\.(jpg|gif|png)$     /var/poze/$2/$1.$2
```

In exemplul de mai sus, prima sub-expresie contine numele fisierului solicitat, iar cea de-a doua extensia. Secventele $1 si $2 din cel de-al doilea argument al directivei se refera la valorile pe care le iau cele doua subexpresii pentru cererea curenta. Atunci cand clientul cere URI-ul */images/pic.jpg*, poza va fi livrata din */var/poze/jpg/pic.jpg*; daca URI-ul este */images/pic1/gif*, va fi livrat continutul fisierului */var/poze/gif/pic1.gif*.

Mecanismul de alias-uri are urmatoarele caracteristici importante:

- clientul nu este constient de operatia efectuata de server, deoarece, ca urmare a cererii efectuate, primeste direct raspunsul ce contine datele dorite. Daca clientul este un browser, continutul barei de adresa nu se va modifica
- alias-urile se proceseaza o singura data – URI-ul nu poate trece succesiv prin mai multe transformari (“alias chaining”)

7.8.3. Redirectionari

O redirectionare reprezinta un raspuns HTTP cu cod 3xx care ii indica clientului, prin intermediul headerului HTTP Location, noua resursa pe care acesta trebuie sa o solicite. Clientul va efectua o noua cerere, iar bara de adresa a browserului (daca clientul este un browser) isi va schimba continutul.

Apache permite redirectionarea clientilor in functie de URI-ul cerut folosind directivele Redirect sau RedirectMatch, cu sintaxele:

```
Redirect            <cod_HTTP>    URI           URL_nou
RedirectMatch       <cod_HTTP>    regex_URI     URL_nou
```

Directivele confrunta URI-ul solicitat de client cu string-ul/regex-ul primit ca argument si, in caz de potrivire, redirectioneaza clientul catre *URL_nou* sau ii trimit codul de raspuns cuprins in cadrul directivei.

Codul HTTP este optional si poate fi:

- unul dintre cuvintele cheie *temp*, *permanent*, *seeother* sau *gone*, care corespund codurilor HTTP 302, 301, 303 respectiv 410
- un cod numeric valid HTTP (ex: 404)
- in cazul in care codul lipseste va fi folosit automat 302

Daca codul este din gama 3xx, prezenta argumentului *URL_nou* este obligatorie, deoarece rezultatul este o redirectionare; pentru alte coduri el este optional:

```
# urmatoarele trei directive sunt echivalente
Redirect     temp   /URIvechi     /URInou
Redirect     302    /URIvechi     /URInou
Redirect            /URI-vechi    /URInou

# trimiterea unui acod de raspuns care nu determina redirectionare
Redirect     404    /inaccesibil_temporar
```

RedirectMatch functioneaza pe aceleasi principii ca *AliasMatch* (prin aplicarea de regex-uri), numai ca determina o redirectionare a clientului.

Atentie! *Directivele Redirect au prioritate fata de eventualele directive Alias cu care se suprapun, indiferent de ordinea in care apar toate aceste directive in fisierul de configurare!*

Nota: *Apache ofera si directivele RedirectPermanent si RedirectTemp, care au efect identic cu Redirect urmat de tipurile respective de redirectionare.*

7.8.4. Reguli complexe de manipulare de URL: mod_rewrite

7.8.4.1. Descriere

Modulul *mod_rewrite* reprezinta cea mai flexibila modalitate de a determina actiunea aplicata unei cereri HTTP. Capabilitatile sale le includ pe cele ale modulelor descrise anterior si le depasesc cu mult – aceasta cu costul cresterii complexitatii de configurare si al consumului de resurse superior.

Pentru a decide ce actiune sa aplice unei cereri, *mod_rewrite* poate tine cont de parametri ai clientului, ai cererii, ai conexiunii, headere HTTP, variabile de mediu etc. Actiunea aplicata poate fi simpla livrare a unei resurse, dar poate fi si o redirectionare sau o masura de control al accesului.

Modulul mod_rewrite introduce 3 directive principale:

- **RewriteEngine** – folosit pentru a activa motorul de rewrite (costisitor si, deci, dezactivat din oficiu). Valori posibile: *On* sau *Off*
- **RewriteRule** - “calul de bataie” al lui *mod_rewrite*, care specifica actiunea aplicata unei cereri si conditiile pentru aplicarea ei
- **RewriteCond** – specifica conditii de aplicare a unui RewriteRule

In afara de acestea, avem directive care regleaza logging-ul (RewriteLog, RewriteLogLevel) si alte cateva directive folosite in scenarii mai speciale (RewriteOptions, RewriteBase etc)

7.8.4.2. Activarea motorului de rescriere

Analiza si rescrierea URL-urilor folosind *mod_rewrite* este dezactivata din oficiu; ea poate fi activata folosind directiva:

```
RewriteEngine       On
```

Directiva poate fi plasata in contextul global, in interiorul host-urilor virtual, sau per-director – fie in interiorul unei directive <Directory>, fie in cadrul unui fisier .htaccess.

Atentie! *Setarile facute in contextul global nu se mostenesc automat in host-urile virtuale! Pentru fiecare virtual host trebuie sa activam separat rescrierea folosind directiva ReweriteEngine!*

7.8.4.3. Directiva RewriteRule

Sintaxa generala a directivei este:

```
RewriteRule  pattern      inlocuitor   [flag-uri]
```

Pattern-ul este un regex care se aplica celei de-a treia parti a URL-ului (numit URI sau URL-Path) – adica ceea ce ramane din URL daca ce eliminam protocolul, host-ul si query string-ul. Spre exemplu, daca scriem in browser *http://www.srv.ro/scripts/1.php?lang=ro*, atunci pattern-ul se va aplica string-ului */scripts/1.php*.

Inlocuitorul poate fi:

- o cale in sistemul de fisiere, de unde serverul trebuie sa livreze clientului resursa dorita
- o cale web (un URI/URL-Path). Avand in vedere ca o cale in sistemul de fisiere si un URI au sintaxa identica, motorul de rewrite va decide daca inlocuitorul este cale in sistemul de fisiere analizand prima portiune ("director") din inlocuitor si verificand daca exista un director cu acel nume in radacina sistemului de fisiere al serverului (/). Daca directorul nu exista, inlocuitorul va fi considerat a fi un URI. Sa consideram exemplul unui server al carui DocumentRoot este stabilit in */var/www* si care contine in fisierul sau de configurare urmatoarea regula de rewrite:

```
RewriteRule  /a     /admin/index.html
```

  Serverul va verifica daca exista directorul */admin* in sistemul sau de fisiere, procedand apoi astfel:
  - ➔ daca directorul exista, serverul va considera inlocuitorul ca fiind o cale absoluta in sistemul sau de fisiere si va livra clientului continutul fisierului */admin/index.html*
  - ➔ in caz contrar, inlocuitorul este considerat a fi un URL-Path – deci va fi prefixat cu valoarea lui DocumentRoot pentru a obtine calea absoluta in sistemul de fisiere: */var/www/admin/index.html*

- un URL absolut, care poate contine si query string:

```
RewriteRule  /mail  http://webmail.server.ro/index.php?from=redirect
```

Inlocuitorul poate folosi valori ale sub-expresiilor din pattern, cu secventele $1...$9, la fel ca in cazul directivelor *AliasMatch*/*RedirectMatch*.

Atentie! *URL-ul este inlocuit* ***integral*** *cu inlocuitor, dupa care poate trece mai departe prin alte reguli de rewrite, acestea aplicandu-si pattern-ul formei deja modificate a URL-ului.*

Flag-urile atasate unei reguli pot influenta modul in care se evalueaza regula sau actiunea aplicata cererii. Iata principalele flag-uri disponibile (pentru lista completa consultati documentatia Apache):

- **nocase** (prescurtat NC) – face pattern-ul case insensitive

- **chain (prescurtat C)** – marcheaza regula curenta ca fiind "conectata" cu urmatoarea, in sensul in care, daca regula curenta nu se aplica (pattern-ul nu corespunde), este sarita si cea urmatoare. Regula urmatoare poate avea la randul sau flag-ul C s.a.m.d.
- **env=numevariabila:valoarevariabila** (prescurtat E) – seteaza variabila dorita in caz de potrivire a regulii
- **forbidden** (prescurtat F) – trimite clientului u nraspuns HTTP cu status 403 (forbidden)
- **gone** (prescurtat G) – analog cu F, dar status 410
- **last** (prescurtat L) – solicita incheierea procesarii URL-ului (echivalentul unui break din programare). Daca mai exista alte reguli dedesubt acestea nu vor mai fi evaluate.
- **passthrough** (prescurtat PT) – URL-ul rezultat din prelucrarile lui mod_rewrite este pasat mai departe altor handlere care il pot evenutal modifica (Alias, ScriptAlias, Redirect etc.). In mod normal URL-ul rezultat nu este accesibil acestora
- **qsappend** (prescurtat QSA) – daca inlocuitorul contine query string, acesta nu il va inlocui pe cel original ci va fi adaugat la acesta
- **redirect<=cod>** (prescurtat R) – trimite clientului un mesaj de redirectionare care URL-ul rezultat din prelucrare. Optional poate fi specificat si codul de raspuns dorit
- **skip=NR** (prescurtat S) – daca regula curenta s-a potrivit (URI-ul a corespuns pattern-ului) atunci vor fi "sarite" urmatoarele *NR* reguli
- **proxy** (prescurtat P) – resursa referita de URL-ul rezultat din prelucrare va fi download-ata de catre Apache (care joaca astfel rolul de client web, prin intermediul modulului sau *mod_proxy*) si predata apoi clientului. Putem astfel sa facem vizibile resurse externe prin intermediul serverului nostru – cu alte cuvinte, Apache va juca un rol de proxy selectiv

Atentie! *Parantezele drepte fac parte din sintaxa de specificare a flag-urilor!*

7.8.4.4. Directiva RewriteCond

Una dintre marile diferente ale modulului *mod_rewrite* fata de *mod_alias* si *mod_redirect* este ca, in afara de a integra actiunile posibile asupra unei cereri (alias/redirect), poate pune o sumedenie de conditii asupra cererii, care sa ia in considerare nu numai URI-ul, ci practic orice alt aspect al cererii adresate de client. In acest scop, o directiva *RewriteRule* poate fi precedata de una sau mai multe directive RewriteCond, ea nefiind aplicata decat daca toate conditiile atasate sunt indeplinite.

```
RewriteCond  sir_testat1  pattern1
RewriteCond  sir_testat2  pattern2
RewriteRule  pattern_A    inlocuitor_A

RewriteCond  sir_testat3  pattern1
RewriteCond  sir_testat4  pattern2
RewriteRule  pattern_B    inlocuitor_B
```

Subliniem doua aspecte:

- un set de conditii *RewriteCond* se aplica din oficiu numai directivei *RewriteRule* ce-i urmeaza, nu tuturor directivelor *RewriteRule* prezente in fisier.
- conditiile atasate unei reguli sunt compuse implicit folosind AND (SI logic) intre fiecare doua conditii consecutive, ceea ce impune ca TOATE conditiile sa fie indeplinite pentru a aplica actiunea cuprinsa in regula. Administratorul poate insa introduce in unele conditii un flag OR care schimba aceasta logica (vezi paragraful despre flag-uri)

Atuni cand exista conditii atasate unei reguli de rewrite, ordinea operatiilor efectuate de catre server este urmatoarea:

- intai se determina daca regula se aplica cererii (!), confruntand URI-ul cererii cu pattern-ul din regula. In caz de nepotrivire, conditiile nu mai sunt evaluate.
- daca URI-ul a corespuns pattern-ului, se evalueaza pe rand conditiile atasate regulii, in ordinea in care apar in fisier. Daca vreuna dintre conditii nu este indeplinita, substitutia specificata in regula nu va mai fi efectuata
- daca evaluarea ajunge la sfarsitul listei de conditii si toate conditiile au fost indeplinite, numai atunci va fi inlocuit URL-ul cu string-ul specificat si se vor aplica eventualele flag-uri

Sintaxa generala a directivei RewriteCond este:

```
RewriteCond  sir_testat   pattern            [flag-uri]
```

Sirul testat in cadrul unui *RewriteCond* poate contine:

- text scris de catre administrator
- back references:
  - valori ale subexpresiilor din pattern-ul regulii de care apartine conditia, folosind $1...$9
  - valori ale subexpresiilor din pattern-ul conditiei precedente, cu %1...%9
- variabile definite automat de catre server, sub forma %{NUME_VARIABILA}. Lista completa a variabilelor poate fi gasita in anexele materialului
- variabile de mediu definite anterior, cu sintaxa %{ENV:ume_variabila}
- variabile SSL, cu sintaxa %{SSL:nume_variabila_ssl}

Pattern-ul aplicat sirului de testat poate fi:

- un regex aplicat sirului testat. Poate fi precedat de ! pentru a impune nepotrivirea intre pattern si sirul testat
- un alt gen de conditie aplicata sirului de testat (poate fi precedata de asemenea de semnul exclamarii pentru a nega sensul conditiei):
  - sirul testat poate fi considerat o cale in sistemul de fisiere, caz in care putem pune conditii asupra existentei, dimensiunii, tipului sau permisiunilor fisierului referit
  - sirul testat poate fi comparat alfabetic cu pattern-ul

Lista de conditii speciale poate fi de asemenea gasita in anexele materialului.

O conditie de rescriere poate avea atasate eventuale flag-uri:

- **nocase** (prescurtat NC) – face compratia intre pattern si sirul testat sa fie case insensitive
- **ornext** (prescurtat OR) – conditia curenta nu va mai fi combinata cu urmatoarea folosind AND, ci OR (SAU logic).

Atentie! *Parantezele drepte fac parte din sintaxa de specificare a flag-urilor!*

```
RewriteCond %{REMOTE_HOST}  ^client1.*  [OR,NC]
RewriteCond %{REMOTE_HOST}  ^client2.*  [OR,NC]
RewriteCond %{REMOTE_HOST}  ^client3.*
RewriteRule .............
```

Exemplu: conexiunile criptate catre un anumit nume si URI al serverului sunt redirectionate noaptea catre o alta adresa:

```
RewriteCond  %{ENV:HTTPS}  !""
RewriteCond  %{HTTP_HOST}  ^www\.srv\.ro$
RewriteCond  %{TIME_HOUR}  >23
RewriteCond  %{TIME_HOUR}  <6
RewriteRule  /search              http://www.yahoo.com       [R]
```

Atentie! *Setarile de rewrite definite in contextul global nu se mostenesc in VH-uri, ci trebuie rescrise acolo (inclusiv RewriteEngine On!) sau mostenite cu* RewriteOptions inherit*!*

Nota: *atunci cand dorim sa luam mai multe decizii ca urmare a indeplinirii unui set de RewriteCond-uri, pentru a nu evalua de fiecare data intregul set de conditii, putem defini o variabila de mediu ca urmare a indeplinirii conditiilor. Aceasta variabila va fi folosita in toate deciziile ulterioare, fara a mai evalua inca o data setul de conditii.*

7.8.4.5. Diagnosticarea operatiilor de rewrite

Avand in vedere complexitatea la care pot ajunge seturile de reguli de rescriere impreuna cu conditiile atasate, ceea ce creste mult posibilitatea de eroare, mod_rewrite ofera posibilitatea consemnarii separate a informatiilor sale de logging. Dispunem in acest scop de doua directive:

- **RewriteLog** – stabileste calea catre fisierul in care vor fi consemnate informatiile de logging legate de operatiile de rewrite
- **RewriteLogLevel** – stabileste gradul de detaliu al informatiilor consemnate. 0 este nivelul minim, 9 cel maxim

```
# in etapa de configurare a serverului putem folosi nivelul maxim, pentru usurarea diagnosticarii
RewriteLog    /var/log/apache/rewrite.log
RewriteLogLevel     9
```

7.9. BIBLIOGRAFIE

- Prezentare HTTP: http://www.jmarshall.com/easy/http/
- RFC-uri
  - HTTP/1.1: http://tools.ietf.org/html/rfc2616
  - Headerele HTTP 1.1: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html
  - Lista headerelor HTTP si RFC de definitie al fiecaruia: http://tools.ietf.org/html/rfc4229
- Lista codurilor de raspuns HTTP: http://en.wikipedia.org/wiki/List_of_HTTP_status_codes
- Documentatia oficiala Apache 2.2: http://httpd.apache.org/docs/2.2
- Loguri Apache: http://httpd.apache.org/docs/2.2/logs.html
- Mecanisme de manipulare URL: http://httpd.apache.org/docs/2.2/urlmapping.html
- Solutii mod_rewrite: http://wiki.apache.org/httpd/Rewrite/
- Carti
  - The Definitive Guide to Apache mod_rewrite (de Rich Bowen) – Apress 2006
  - Apache Security (de Ivan Ristic) – O'Reilly 2005
  - Pro Apache, 3rd Edition (de Peter Wainwright) - APress 2004
  - Apache Cookbook, 2nd Edition (de Rich Bowen) – O'Reilly 2008

7.10. ANEXA 1 – tabela descriptori informatii logging

Format String	Description
%%	The percent sign
%a	Remote IP-address
%A	Local IP-address
%B	Size of response in bytes, excluding HTTP headers.
%b	Size of response in bytes, excluding HTTP headers. In CLF format, i.e. a '-' rather than a 0 when no bytes are sent.
%{Foobar}C	The contents of cookie Foobar in the request sent to the server.
%D	The time taken to serve the request, in microseconds.
%{FOOBAR}e	The contents of the environment variable FOOBAR
%f	Filename
%h	Remote host
%H	The request protocol
%{Foobar}i	The contents of Foobar: header line(s) in the request sent to the server. Changes made by other modules (e.g. mod_headers) affect this.
%k	Number of keepalive requests handled on this connection. Interesting if KeepAlive is being used, so that, for example, a '1' means the first keepalive request after the initial one, '2' the second, etc...; otherwise this is always 0 (indicating the initial request). Available in versions 2.2.11 and later.
%l	Remote logname (from identd, if supplied). This will return a dash unless mod_ident is present and IdentityCheck is set On.
%m	The request method
%{Foobar}n	The contents of note Foobar from another module.
%{Foobar}o	The contents of Foobar: header line(s) in the reply.
%p	The canonical port of the server serving the request
%{format}p	The canonical port of the server serving the request or the server's actual port or the client's actual port. Valid formats are canonical, local, or remote.
%P	The process ID of the child that serviced the request.
%{format}P	The process ID or thread id of the child that serviced the request. Valid formats are pid, tid, and hextid. hextid requires APR 1.2.0 or higher.
%q	The query string (prepended with a ? if a query string exists, otherwise an empty string)
%r	First line of request
%R	The handler generating the response (if any).
%s	Status. For requests that got internally redirected, this is the status of the *original* request --- %>s for the last.
%t	Time the request was received (standard english format)
%{format}t	The time, in the form given by format, which should be in strftime(3) format. (potentially localized)
%T	The time taken to serve the request, in seconds.
%u	Remote user (from auth; may be bogus if return status (%s) is 401)

%U	The URL path requested, not including any query string.
%v	The canonical ServerName of the server serving the request.
%V	The server name according to the UseCanonicalName setting.
%X	Connection status when response is completed: X = connection aborted before the response completed. + = connection may be kept alive after the response is sent. – = connection will be closed after the response is sent. – (This directive was %c in late versions of Apache 1.3, but this conflicted with the historical ssl %{var}c syntax.)
%I	Bytes received, including request and headers, cannot be zero. You need to enable mod_logio to use this.
%O	Bytes sent, including headers, cannot be zero. You need to enable mod_logio to use this.

7.11. ANEXA 2 – variabile de mediu definite de mod_ssl

Variable Name:	Value Type:	Description:
HTTPS	flag	HTTPS is being used.
SSL_PROTOCOL	string	The SSL protocol version (SSLv2, SSLv3, TLSv1)
SSL_SESSION_ID	string	The hex-encoded SSL session id
SSL_CIPHER	string	The cipher specification name
SSL_CIPHER_EXPORT	string	true if cipher is an export cipher
SSL_CIPHER_USEKEYSIZE	number	Number of cipher bits (actually used)
SSL_CIPHER_ALGKEYSIZE	number	Number of cipher bits (possible)
SSL_COMPRESS_METHOD	string	SSL compression method negotiated
SSL_VERSION_INTERFACE	string	The mod_ssl program version
SSL_VERSION_LIBRARY	string	The OpenSSL program version
SSL_CLIENT_M_VERSION	string	The version of the client certificate
SSL_CLIENT_M_SERIAL	string	The serial of the client certificate
SSL_CLIENT_S_DN	string	Subject DN in client's certificate
SSL_CLIENT_S_DN_x509	string	Component of client's Subject DN
SSL_CLIENT_I_DN	string	Issuer DN of client's certificate
SSL_CLIENT_I_DN_x509	string	Component of client's Issuer DN
SSL_CLIENT_V_START	string	Validity of client's certificate (start time)
SSL_CLIENT_V_END	string	Validity of client's certificate (end time)
SSL_CLIENT_V_REMAIN	string	Number of days until client's certificate expires
SSL_CLIENT_A_SIG	string	Algorithm used for the signature of client's certificate
SSL_CLIENT_A_KEY	string	Algorithm used for the public key of client's certificate
SSL_CLIENT_CERT	string	PEM-encoded client certificate
SSL_CLIENT_CERT_CHAIN_n	string	PEM-encoded certificates in client certificate chain
SSL_CLIENT_VERIFY	string	NONE, SUCCESS, GENEROUS or FAILED:reason
SSL_SERVER_M_VERSION	string	The version of the server certificate
SSL_SERVER_M_SERIAL	string	The serial of the server certificate
SSL_SERVER_S_DN	string	Subject DN in server's certificate
SSL_SERVER_S_DN_x509	string	Component of server's Subject DN
SSL_SERVER_I_DN	string	Issuer DN of server's certificate
SSL_SERVER_I_DN_x509	string	Component of server's Issuer DN
SSL_SERVER_V_START	string	Validity of server's certificate (start time)
SSL_SERVER_V_END	string	Validity of server's certificate (end time)
SSL_SERVER_A_SIG	string	Algorithm used for the signature of server's certificate
SSL_SERVER_A_KEY	string	Algorithm used for the public key of server's certificate
SSL_SERVER_CERT	string	PEM-encoded server certificate

7.12. ANEXA 3 – variabile de server disponibile lui RewriteCond

HTTP headers	connection & request	date and time	server internals	specials
HTTP_USER_AGENT	REMOTE_ADDR	TIME_YEAR	DOCUMENT_ROOT	API_VERSION
HTTP_REFERER	REMOTE_HOST	TIME_MON	SERVER_ADMIN	THE_REQUEST
HTTP_COOKIE	REMOTE_PORT	TIME_DAY	SERVER_NAME	REQUEST_URI
HTTP_FORWARDED	REMOTE_USER	TIME_HOUR	SERVER_ADDR	REQUEST_FILENAME
HTTP_HOST	REMOTE_IDENT	TIME_MIN	SERVER_PORT	IS_SUBREQ
HTTP_PROXY_CONNECTION	REQUEST_METHOD	TIME_SEC	SERVER_PROTOCOL	HTTPS
HTTP_ACCEPT	SCRIPT_FILENAME	TIME_WDAY	SERVER_SOFTWARE	
	PATH_INFO	TIME		
	QUERY_STRING			
	AUTH_TYPE			

7.13. ANEXA 4 – pattern-uri speciale RewriteCond

<CondPattern	Treats the CondPattern as a plain string and compares it lexicographically to TestString. True if TestString
>CondPattern	Treats the CondPattern as a plain string and compares it lexicographically to TestString. True if TestString lexicographically follows CondPattern.
=CondPattern	Treats the CondPattern as a plain string and compares it lexicographically to TestString. True if TestString is lexicographically equal to CondPattern (the two strings are exactly equal, character for character). If CondPattern is "" (two quotation marks) this compares TestString to the empty string.
-d	Treats the TestString as a pathname and tests whether or not it exists, and is a directory.
-f	Treats the TestString as a pathname and tests whether or not it exists, and is a regular file.
-s	Treats the TestString as a pathname and tests whether or not it exists, and is a regular file with size greater than zero.
-l	Treats the TestString as a pathname and tests whether or not it exists, and is a symbolic link.
-x	Treats the TestString as a pathname and tests whether or not it exists, and has executable permissions. These permissions are determined according to the underlying OS.
-F	Checks whether or not TestString is a valid file, accessible via all the server's currently-configured access controls for that path. This uses an internal subrequest to do the check, so use it with care - it can impact your server's performance!
-U	Checks whether or not TestString is a valid URL, accessible via all the server's currently-configured access controls for that path. This uses an internal subrequest to do the check, so use it with care - it can impact your server's performance!