8. SERVERUL PROXY SQUID

8.1. Principii generale

8.1.1. Ce este un proxy web si care sunt beneficiile sale

La modul general, un server proxy este un soft care intermediaza cererile clientilor catre alte servere: in loc ca clientul sa solicite resursa dorita direct de pe serverul de origine, cererea este adresata serverului proxy care mai apoi o directioneaza catre serverul potrivit. Softurile de tip proxy actioneaza la nivelul aplicatie, ceea ce inseamna ca "inteleg" si prelucreaza datele protocoalelor de nivel aplicatie; spre exemplu, un proxy SMTP va distinge intre diversele tipuri de cereri SMTP si le va putea aplica eventuale reguli de acces. O alta consecinta a functionarii la nivel aplicatie este ca un soft de proxy nu functioneaza pentru toate protocoalele, ci numai pentru cele pe care le intelege; de aceea vorbim de proxy web, proxy SMTP etc.

Vom numi in continuare "servere de origine" serverele la care se conecteaza proxy-ul in numele clientilor (cele de pe care provine de fapt informatia).

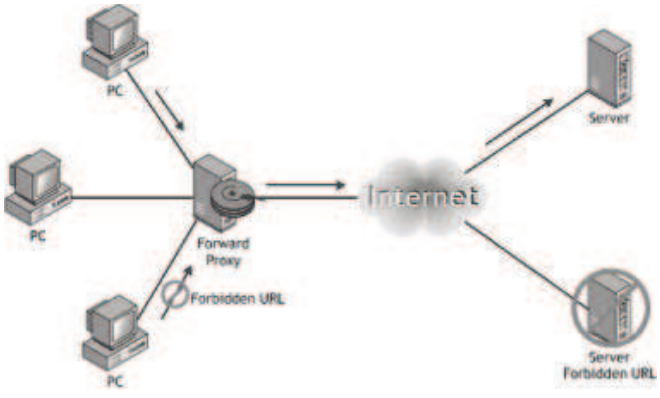
Un proxy web este un soft construit sa inteleaga si sa proceseze cereri HTTP. In cadrul unei cereri primite de la client, softul de proxy poarta doua dialoguri independente:

- unul cu clientul, in care acesta solicita resursele dorite si primeste raspunsurile de la serverul proxy
- unul cu serverul de origine, in care serverul proxy joaca rolul de client, solicitand informatiile necesare

Iata cateva dintre beneficiile sau posibilele utilizari ale unui server proxy web:

- **caching**. Serverul poate pastra local copii ale informatiilor download-ate de la serverele de origine; cu cat exista mai multi clienti ai serverului proxy, cu atat se manifesta mai pregnant urmatoarele implicatii:
  - creste probabilitatea ca o anumita resursa sa existe deja in cache, ceea ce inseamna ca multi clienti vor primi datele dorite direct din cache, asadar la viteza in general mult superioara comparativ cu conectarea directa la serverul de origine. Efectul observat de client este incarcarea mai rapida a paginilor
  - cu cat sunt refolosite mai intens resursele din cache-ul proxy-ului, cu atat scade traficul cu serverele de origine. Acest fapt poate fi folosit atunci cand traficul in afara unei retele este costisitor sau banda trebuie sa fie disponibila in alte scopuri
- **controlul accesului**. Serverul proxy poate permite accesul selectiv al clientilor la resursele web, fie bazandu-se pe caracteristicile acestora (IP, nume DNS etc), fie solicitandu-le autentificare
- **filtrare de continut**. Deoarece serverul proxy "vorbeste HTTP", el are acces la toti parametrii cererilor si raspunsurilor HTTP si poate filtra mesajele in functie de criterii complexe. De asemenea, softul proxy poate interfata cu softuri externe de filtrare de continut (ex: antivirus, anti-malware etc)
- **urmarire trafic**. Profitand de rolul sau de intermediar, serverul proxy poate consemna in jurnale traficul web ce-l parcurge in scopul monitorizarii sau al alcatuirii de statistici
- **anonimizare.** Avand in vedere ca, din punct de vedere al serverului de origine, proxy-ul este clientul, serverele proxy pot fi folosite si pentru ascunderea identitatii reale a celui care solicita resursele (clientul serverului proxy)
- **securizare**. Un proxy poate fi folosit pe post de application-level firewall, fiind plasat in fata serverului real si efectuand filtrari ale mesajelor destinate acestuia
- **load balancing**. Un server proxy poate fi configurat sa primeasca toate cererile adresate unui site si apoi sa le directioneze catre mai multe servere web care gazduiesc aceeasi informatie, in scopul distribuirii incarcarii

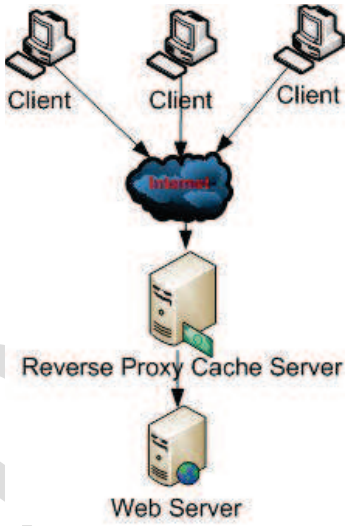
8.1.2. Moduri de functionare ale unui proxy



Din punct de vedere al scopului sau, un proxy poate fi configurat in doua moduri:

- **forward proxy**. Acest tip de proxy este destinat sporirii vitezei/reducerii traficului web pentru un numar de clienti (spre exemplu, cei care formeaza o retea locala a unei firme). El este plasat de obicei in vecinatatea clientilor – deseori, in aceeasi retea locala. Clientii solicita resurse proxy-ului si acesta le cere la randul sau de la serverul de origine, oricare ar fi acesta. Serverul proxy poate mentine

un cache cu informatie (reducand astfel utilizarea conexiunii internet) si poate aplica filtre asupra continutului ce-l tranziteaza. Un forward proxy are urmatoarele caracteristici:
  - clientii trebuie de obicei configurati sa foloseasca serverul proxy (cu exceptia proxy-ului transparent – vezi mai jos)
  - proxy-ul ofera acces la informatii aflate pe mai multe servere de origine
  - proxy-ul poate solicita clientilor autentificare inainte de a actiona in numele lor
- **reverse proxy**. Acest tip de proxy este plasat in vecinatatea unui server web si, din punct de vedere al clientului, el ESTE serverul. Clientii ii adreseaza cereri, insa el le paseaza mai departe unuia sau mai multor servere de origine care i-au fost preconfigurate. Caracteristicile unui reverse proxy sunt:
  - nu presupune configurari speciale din partea clientilor, deoarece acestia traiesc cu impresia ca dialogheaza chiar cu serverul de origine
  - numarul de servere de origine este limitat in comparatie cu cazul forward proxy, deoarece scopul proxy-ului este acum optimizare/filtrare a accesului la unul sau cateva site-uri particulare
  - proxy-ul nu va solicita in general autentificare clientilor, deoarece el intermediaza accesul la informatie publica



Nota: *termenul de "vecinatate" de mai devreme nu trebuie luat in sens geografic, ci de RTT (Round Trip Time). Clientul poate beneficia pe deplin de serviciile unui forward proxy doar in masura in care timpul de acces la el si banda disponibila sunt superioare celor cu serverele de origine.*

Din punct de vedere al modului de configurare a clientilor in scopul utilizarii serverului proxy, distingem:
- **server proxy standard**. Acest tip de proxy mentine dialoguri distincte cu clientii si cu serverele de origine. Clientii trebuie configurati sa foloseasca serverul proxy; cererile ii sunt adresate explicit acestuia.
- **server proxy transparent**. Un astfel de proxy intercepteaza cererile clientilor in drumul lor catre serverul de origine, jucand in mod fortat rolul de proxy, fara acceptul clientului. Nu este nevoie de nicio configurare speciala din partea clientilor, deoarece acestia nu sunt constienti de deturnarea traficului lor si isi adreseaza cererile direct serverului de origine. Trebuie insa prevazuta o modalitate de a redirectiona traficul web catre softul de proxy (ex: o forma de NAT)

8.1.3. Mecanisme de control al caching-ului in HTTP

8.1.3.1. Principii

Un proxy server care ofera si serviciul de caching detine copii ale resurselor obtinute de pe serverele web de origine. Acestea pot fi livrate clientilor la cererile ulterioare – fara a re-contacta serverul de origine – numai in masura in care proxy-ul poate fi sigur de conformitatea resursei detinute cu originalul.

Specificatia HTTP ofera urmatoarele mecanisme pentru evitarea re-downloadarii unei resurse:
- **expirare**. Fiecare resursa memorata in cache poate avea un timp de viata atasat; cat timp acesta nu a expirat, proxy-ul poate oferi resursa clientilor fara a trebui sa mai contacteze serverul ei de origine. Timpul de viata poate fi determinat/limitat de server sau de client; spre exemplu, serverul poate indica proxy-ului, prin intermediul unor headere HTTP, momentul expirarii resursei sau timpul ei de viata
- **validare**. Dupa ce timpul de viata al unei resurse s-a scurs, proxy-ul poate verifica conformitatea sa cu originalul fara a o re-descarca de pe serverul de origine prin folosirea unor cereri HTTP conditionale, in urma carora serverul poate semnaliza ca respectiva resursa nu s-a modificat intre timp
- **invalidare**. Atunci cand proxy-ul primeste o cerere adresata unei resurse aflate in cache dar care produce modificari asupra resursei (ex: o cerere POST, PUT sau DELETE) copia din cache este invalidata, fiind necesara re-verificarea conformitatii ei cu originalul

Fiecare dintre cele 3 entitati implicate – server de origine, proxy si client – poate controla sau influenta caching-ul, dupa cum urmeaza:
- serverul de origine este cel mai in masura sa indice ce informatii se preteaza la a fi memorate in cache. Spre exemplu, paginile web generate dinamic (folosind PHP, ASP, Perl etc) pot sa nu aiba de fiecare data acelasi continut si de aceea nu sunt in general introduse in cache

- serverul proxy poate fi configurat sa treaca peste indicatiile serverului si sa foloseasca propria strategie de caching (chiar si abatandu-se de la specificatia HTTP!)
- clientul poate indica la randul sau proxy-ului ca doreste re-descarcarea/revalidarea unei resurse, chiar si in conditiile in care timpul de viata al acesteia nu s-a scurs inca

Nota: *fiecare resursa din cache are atasat URL-ul de provenienta. De aceea, pentru a profita intr-adevar de caching, trebuie ca de fiecare data ea sa fie accesata prin intermediul aceluiasi URL.*

8.1.3.2. Headere HTTP pentru controlul caching-ului si utilizarea lor

Exista cel putin doua elemente ale lantului de transmisiune web care pot efectua caching:

- serverul proxy. El este ceea ce se numeste un "shared cache", deoarece de informatiile pe care le memoreaza pot beneficia mai multi clienti
- browser-ul client. Acesta este un "private cache" deoarece informatiile pe care le pastreaza vor avantaja un singur user

Prezentam in continuare principalele headere HTTP folosite in controlul caching-ului:

- **Last-Modified** – prin intermediul acestui header, serverul web indica data ultimei modificari a unei resurse. Pe baza ei, serverul proxy poate fi configurat sa seteze momentul expirarii atunci cand nicio alta informatie nu este disponibila
- **Expires** – folosind acest header, serverul web transmite momentul explicit al expirarii unei resurse
- **Cache-Control** – poate fi folosit atat de catre server, cat si de catre client pentru controlul caching-ului. Valoarea sa este compusa din directive individuale, printre care se numara:
  - *public* – informatia este public disponibila si deci poate fi memorata in cache de catre un server proxy, deoarece in acest fel ar beneficia mai multi useri de pe urma ei
  - *private* – informatia este destinata unui singur user si deci nu trebuie memorata in cache-ul unui proxy; in schimb browserul are permisiunea de a pastra in cache informatia
  - *no-cache* – serverul proxy poate memora in cache informatia insa este fortat sa o valideze de fiecare data inainte de a o trimite unui client, indiferent de vechimea sa
  - *no-store* – informatia nu trebuie memorata in niciun cache (nici shared, nici privat)
  - *max-age=nrsecunde* – impune timpul de viata al resursei in cache. Spre deosebire de Expire, aceasta directiva stabileste momentul expirarii relativ la momentul curent
  - *s-maxage=nrsecunde* – similar cu max-age, dar aplicabil numai proxy-urilor, nu si cache-urilor de browser
- **ETag** – este un header folosit pentru verificarea conformitatii unei resurse din cache cu originalul. Serverul poate atasa fiecarei resurse livrate un identificator, care se schimba la fiecare modificare a resursei. Proxy-urile memoreaza etag-ul impreuna cu resursa si, ori de cate ori e nevoie de validare, verifica daca pe server resursa in cauza are acelasi etag. O modalitate convenabila de calcul al etag-ului se bazeaza pe algoritmii de hashing
- headere pentru cereri conditionale. Aceste headere atasabile unei cereri sunt folosite de catre proxy pentru a-i indica serverului ca doreste sa re-descarce resursa respectiva numai in cazul in care ea nu s-a modificat intre timp. Serverul va raspunde cu 304 (Not Modified) in cazul in care aceasta a ramas neschimbata sau cu insasi resursa in cazul modificarii. Conditia de nemodificare a resursei poate lua doua forme in cererea adresata de proxy serverului de origine:
  - specificand in cerere data ultimei modificari cunoscuta de proxy si lasand serverul sa o compare cu data ultimei modificari a aceleiasi resurse. Header folosit: **If-Modified-Since.** Resursa va fi livrata proxy-ului numai daca s-a modificat ulterior datei specificate, in caz contrar serverul returnand cod 304
  - specificand in cerere Etag-ul si lasand serverul sa-l compare cu cel curent al resursei dorite. Header folosit: **If-Match**. Resursa va fi livrata numai daca ETag-ul curent difera de cel specificat de proxy

8.1.4. Masurarea performantelor de caching ale unui proxy

Exista doua moduri de a evalua eficienta caching-ului efectuat de un server proxy:

- **hit rate** – reprezinta raportul intre numarul de cereri livrate din cache si numarul total de cereri. Un hit rate mare inseamna ca multe cereri sunt servite din cache, cu avantajul vitezei sporite de incarcare a paginilor din punct de vedere al clientilor

- **byte hit rate** – reprezinta raportul intre numarul total de octeti livrati din cache si numarul total de octeti livrati de proxy. A maximiza byte hit rate-ul inseamna a minimiza traficul proxy-ului cu serverele de origine, dar cu posibilul dezavantaj al scaderii hit rate-ului

Cele doua valori pot diferi destul de mult, ba chiar pot "trage" in directii diferite, totul depinzand de structura traficului ce tranziteaza serverul:

- daca traficul este format preponderent din informatii de mari dimensiuni si proxy-ul le depune in cache, un numar relativ mic de obiecte va epuiza cache-ul. Cat timp obiectele memorate in cache sunt "populare" (solicitate des), byte hit rate-ul va fi unul foarte bun. Pe de alta parte, daca obiectele sunt mari, cache-ul va putea contine putine obiecte si deci hit rate-ul ar putea scadea. Sa consideram exemplul extrem al unui obiect care ocupa tot cache-ul: ori de cate ori este solicitat, numarul de octeti livrati din cache va fi mare si deci byte hit rate-ul va creste; pe de alta parte, din cauza acestui obiect nu mai pot patrunde in cache altele si deci hit rate-ul va scadea.
- daca traficul este format preponderent din obiecte mici si acestea sunt memorate in cache, efectul este cresterea hit rate-ului, caci in aceeasi cantitate de cache incap acum mai multe obiecte, si deci probabilitatea ca un obiecte solicitat de client sa se regaseasca in cache este mai mare

8.2. Prezentare si configurare de baza

8.2.1. Fisa serverului

Iata cateva elemente care trebuie cunoscute inainte de a incepe configurarea squid:

- executabil: **squid**, sau in unele distributii **squid3** (ex: Ubuntu)
- fisier de configurare implicit: **/etc/squid/squid.conf** sau **/etc/squid3/squid.conf**
- oprire: **squid -k shutdown**
- reconfigurare din mers: **squid -k reconfigure**
- validarea fisierului de configurare: **squid -k parse**

Fisierul de configurare contine un ansamblu de directive de forma *numeDirectiva parametri*. Comentariile incep cu #.

8.2.2. Configurare de baza

Configurarea de pornire a serverului *squid* presupune aspecte precum:

- specificarea adresei/portului pe care va actiona serverul. Implicit serverul asculta pe toate adresele definite in sistem, pe portul 3128
- specificarea modului de actiune – forward proxy, reverse proxy, transparent proxy
- userul si grupul cu care vor rula procesele squid
- (optional) stabilirea informatiilor care apar pe paginile de eroare

Directivele de configurare corespunzatoare sunt urmatoarele:

```
# port
http_port 1234
#...posibil si: http_port 10.0.0.100:1234 (pentru a asculta pe o anumita adresa locala)

# user si group neprivilegiat pentru rularea proceselor squid
cache_effective_user squid
cache_effective_group squid

# numele statiei si mailul administratorului, asa cum apar ele pe paginile de eroare
visible_hostname proxy.example.org
cache_mgr admin@example.org
```

Modul de actiune al serverului se stabileste, incepand cu versiunea 2.6, tot cu ajutorul directivei *http_port*, pasandu-i parametri suplimentari. Acestia vor fi detaliati in sectiunile corespunzatoare (proxy transparent, reverse proxy etc)

8.3. Configurarea caching-ului

8.3.1. Configurarea caching-ului pe hard-disk

8.3.1.1. Elemente de configurare

Configurarea caching-ului pe hard-disk implica urmatoarele aspecte:

- stabilirea directorului de caching si a parametrilor de gestionare a acestuia, folosind directiva **cache_dir**
- stabilirea limitelor de ocupare ale acestui director, folosind directivele **cache_swap_low** si **cache_swap_high**
- stabilirea dimensiunilor pentru informatiile ("obiectele") introduse in cache, folosind **minimum_object_size** si **maximum_object_size**
- stabilirea strategiei de inlocuire a obiectelor din cache folosind **cache_replacement_policy**

8.3.1.2. Spatiul de stocare

Formatul general al directivei *cache_dir* este urmatorul:

```
cache_dir sistemStocare directorCache dimensiuneMaximaInMB nrDirectoareNivel1 nrDirectoareNivel2
```

Sistemul de stocare indica strategia de memorare a informatiilor in cache (structura de directoare, modalitate de gestionare a proceselor/firelor de executie astfel incat scrierea pe hard-disk sa nu devina veriga slaba a serverului in cazul unui proxy cu activitate intensa). Ufs reprezinta sistemul traditional, matur, insa exista si cateva solutii mai recente, activabile la compilare (diskd, aufs, coss).

In directorul de cache vor fi create doua niveluri de directoare. Imediat sub *directorCache* vor fi create *nrDirectoareNivel1* subdirectoare, iar in fiecare dintre acestea cate *nrDirectoareNivel2* subdirectoare. Totalul datelor memorate in aceste directoare nu poate depasi dimensiunea maxima specificata in MB.

```
cache_dir ufs /var/spool/squid 1000 10 100
```

Odata descris cu ajutorul directivei *cache_dir*, spatiul de stocare trebuie initializat folosind comanda *squid -z*. Aceasta are ca efect crearea tuturor directoarelor de pe cele doua niveluri, cu permisiunile potrivite. **Atentie! Directorul de cache si toate subdirectoarele sale vor fi create de catre un proces squid ce ruleaza cu UID-ul cache_effective_user si grupul cache_effective_group!**

```
# squid -z
2011/03/04 12:48:50| Creating Swap Directories
2011/03/04 12:48:50| Making directories in /var/spool/squid/00
2011/03/04 12:48:50| Making directories in /var/spool/squid/01
[...restul output-ului a fost omis...]
```

8.3.1.3. Controlul utilizarii cache-ului

In cache nu sunt memorate decat obiectele a caror dimensiune este situata intre **minimum_object_size** si **maximum_object_size**. Valorile ambelor directive sunt exprimate in KB; minimul implicit este 0, ceea ce inseamna ca vor fi memorate in cache obiecte oricat de mici, iar maximul implicit este de 4 MB. A stabili un maxim ridicat inseamna memorarea in cache de obiecte de dimensiuni mari, ceea ce salveaza banda prin refolosirea lor; aceste obiecte tind insa sa fie putine si sa ocupe spatiu mult, ceea ce inseamna o defavorizare a obiectelor mici si deci o scadere de viteza in cazul lor.

```
minimum_object_size 4 KB
maximum_object_size 8192 KB
```

Directiva **cache_swap_low** stabileste *procentul* de spatiu ocupat (din totalul stabilit prin *cache_dir*) de la care incepe inlocuirea obiectelor din cache. Cat timp spatiul ocupat nu depaseste aceasta valoare, squid nu sterge informatie din cache; odata pragul depasit, inlocuirea devine din ce in ce mai agresiva pe masura ce procentul se apropie de **cache_swap_high.** Squid incearca sa mentina procentul de utilizare cat mai apropiat de **cache_swap_low**.

```
# valorile default:
cache_swap_low 90   // 90%
cache_swap_high 95  // 95%
```

Atunci cand trebuie eliminata informatie din cache, Squid decide ce obiecte sa stearga pe baza strategiei configurate cu ajutorul directivei **cache_replacement_policy**. Valoarea ei poate fi una dintre urmatoarele:

- **lru** (Least Recently Used) – squid va sterge obiectele mai vechi, pastrandu-le pe cele recente
- **heap GDSF** (Greedy-Dual Size Frequency) – aceasta strategie de stergere favorizeaza obiectele mici, pastrandu-le in cache si astfel maximizand hit rate-ul. In schimb, daca exista obiecte mari solicitate des, acestea vor fi defavorizate si deci byte hit rate-ul va scadea
- **heap LFUDA** (Least Frequently Used with Dynamic Aging) – o strategie care favorizeaza obiectele populare (solicitate des), indiferent de dimensiunea lor. Aceasta abordare maximizeaza byte hit rate-ul, posibil in detrimentul hit rate-ului (caci cache-ul va putea fi epuizat de un numar mult mai mic de obiecte)

Nota: *pentru a maximiza intr-adevar byte hit rate-ul in cazul LFUDA este de obicei necesara si cresterea valorii lui* ***maximum_object_size****!*

```
cache_replacement_policy heap LFUDA
maximum_object_size 100000 KB
```

8.3.2. Configurarea caching-ului in memorie

Squid dispune si de un cache pastrat in memoria RAM; acesta este folosit pentru a memora informatii precum:

- obiectele aflate in curs de transferare de pe serverele de origine (“in-transit objects”)
- obiectele des solicitate de clienti (“hot objects”)
- obiectele inexistente (“negative-cached objects”) – squid salveaza temporar in memorie - si refoloseste! - rezultatele negative obtinute de la serverele de origine (ex: resursa inexistenta (cod HTTP 404), conexiune refuzata de server etc)

Obiectele in-transit au prioritatea cea mai mare – celelalte doua tipuri de obiecte sunt sterse pentru a le face loc in caz de nevoie.

Controlul caching-ului in RAM se realizeaza prin intermediul urmatoarelor directive:

- **cache_mem** – stabileste cantitatea maxima de memorie pe care squid o dedica celor trei tipuri de obiecte prezentate. Valoarea default este de 8 MB, insa poate fi crescuta in limita memoriei RAM disponibile

  Atentie! Directiva cache_mem nu stabileste cantitatea maxima de memorie ocupata de procesul squid! *Pe langa cele amintite, squid mentine in RAM meta-informatie despre obiectele din cache si multe alte informatii necesare functionarii!*

- **maximum_object_size_in_memory** – ca si in cazul memorarii pe hard-disk, exista o limita maxima (nu insa si una minima!) a dimensiunii obiectelor memorate in cache-ul din RAM
- **memory_replacement_policy** – acelasi efect (si aceleasi valori posibile) ca directiva *cache_replacement_policy* in cazul memorarii pe hard-disk

8.4. Logging, depanare, administrare

8.4.1. Controlul logging-ului

Squid poate mentine multiple loguri, dupa cum urmeaza:

- logul de functionare a serverului, a carui locatie se stabileste cu directiva **cache_log**. Aici sunt consemnate mesajele legate de diferitele operatii efectuate de server; gradul de detaliu este reglabil cu ajutorul directivei *debug_options*
- logul de cereri, stabilit cu directiva **access_log**. Acesta este fisierul in care se consemneaza toate cererile primite de squid si rezultatele lor (asemanator cu access log-ul din Apache)
- logul de activitate al obiectelor din cache, stabilit cu directiva **cache_store_log**. Aici se consemneaza introducerea si stergerea din cache a fiecarui obiect impreuna cu parametrii operatiei in cauza
- loguri optionale. Acestea trebuie in primul rand activate la compilarea serverului squid (prin folosirea unei optiuni speciale pasate scriptului *configure*) si apoi configurate folosind directive specifice:
  - logul de browsere, stabilit cu **useragent_log**. In acest log se memoreaza valoarea headerului HTTP *User-Agent* pentru fiecare cerere primita de la clienti
  - logul de referrer – stabilit cu **referrer_log**. In acest log se memoreaza valoarea headerului HTTP *Referer* pentru fiecare cerere primita de la clienti

Fiecare dintre directivele mentionate primeste ca valoare o cale catre un fisier si, optional, un format de log si unul sau mai multe acl-uri. Atunci cand se doreste anularea log-ului in cauza se va folosi *none* pe post de cale catre fisierul log.

```
cache_log  /var/log/squid/cache.log
access_log /var/log/squid/access.log  format1  retea1  retea2
```

Formatul de logging poate fi stabilit cu ajutorul directivei **logformat**, intr-un mod asemanator cu cel al directivei *LogFormat* din Apache.

Eventualele ACL-uri au rolul de filtrare a cererilor care produc informatii in log. **Vor fi consemnate in log numai cererile care respecta TOATE acl-urile specificate!**

Pentru ca logurile sa nu creasca peste masura, *squid* ofera posibilitatea rotirii acestora cu comanda:

```
# squid -k rotate
```

8.4.2. Modalitati de depanare a configuratiei serverului

Administratorul de squid are la dispozitie multiple modalitati de a primi feedback de la serverul sau:

- verificarea validitatii fisierului de configurare:

```
# squid -k parse
```

- pornirea serverului in foreground, cu informatii de logging detaliate afisate pe ecran:

```
# squid -Nd1
```

- reglarea gradului de detaliu al informatiilor consemnate in *cache_log*, folosind directiva **debug_options**. Aceasta primeste ca parametri una sau mai multe perechi de forma *sectiune,nivel.* Sectiunea precizeaza categoria de operatii pentru care se doreste schimbarea nivelului; valorile posibile sunt intre 0 si 93, iar lista sectiunilor posibile este disponibila online (vezi bibliografie). Nivelul este un intreg cuprins 0-9:

```
# grad maxim de detaliu pentru operatiile de aplicare a ACL-urilor
debug_options  ALL,1  28,9
```

- utilizarea informatiilor furnizate de cache manager (vezi sectiunile urmatoare)

8.4.3. Facilitati speciale ale serverului squid

Serverul squid ofera doua facilitati speciale, de mare folos pentru administratorul de server:

- posibilitatea de a obtine, direct de la server, informatii ce tin de functionarea acestuia (statistici, elemente de configurare, lista de obiecte din cache sau memorie etc). Aceasta facilitate va fi tratata intr-o sectiune dedicata
- posibilitatea de a sterge fortat obiecte din cache. Acest lucru este posibil prin introducerea de catre squid a unui nou tip de cerere HTTP, si anume PURGE; acest tip de cerere este inteles de catre squid insa nu si de catre un server web normal. Clientii pot adresa squid-ului o cerere PURGE insotita de URL-ul original al resursei pentru a o sterge din cache

Aceste servicii suplimentare sunt disponibile clientilor tot prin intermediul cererilor HTTP adresate serverului - nu este nevoie nici de analiza de loguri, nici de acces low-level la directorul de cache.

8.4.4. Utilitarul squidclient

Utilitarul *squidclient* este un client HTTP care permite formularea oricarui fel de cerere HTTP catre serverul proxy. Motivul pentru care am folosi *squidclient* in loc de alti clienti (*wget*, *curl* sau un browser) este ca el suporta (si permite accesul la) facilitatile speciale ale serverului squid mentionate anterior.

Sintaxa generala de apelare este:

```
squidclient optiuni URL
```

unde optiunile definesc diversele caracteristici dorite ale cererii, iar URL-ul este cel solicitat proxy-ului.

Iata cateva exemple comentate de utilizare:

```
# conectare la squid ruland pe alta statie decat cea locala si alt port decat cel default
squidclient -h 10.0.0.100 -p 8888 http://infoacademy.net

# obtinerea unei resurse autentificate HTTP (a nu se confunda cu autentificarea de proxy!)
squidclient -U username -W parola http://site.org/private/index.php

# obtinerea unei resurse folosind squid local care solicita autentificare
# Atentie! Optiunile -u si -w folosesc litere mici de aceasta data!
squidclient -u user -w pass http://example.com

# stergerea unui obiect din cache; -m stabileste tipul de cerere HTTP ("method")
squidclient -m PURGE http://infoacademy.net/poze/1.jpg

# accesare lista de categorii cache manager
squidclient mgr:menu
```

Operatia de stergere din cache solicita o atentie speciala, deoarece este distructiva si in consecinta trebuie permisa numai in mod controlat. Iata un exemplu de set de directive care, adaugate in *squid.conf*, permit stergerea numai de pe statia pe care ruleaza suiqd:

```
acl localhost src 127.0.0.1
acl stergere method PURGE
http_access allow stergere localhost
http_access deny stergere
```

8.4.5. Cache manager-ul

Cache manager-ul este o interfata squid care permite ca, prin cereri HTTP adresate serverului, administratorul sa obtina informatii detaliate despre functionarea acestuia. Cererile adresate cache manager-ului se disting prin faptul ca protocolul din cadrul URL-ului solicitat este **cache_object://** (in loc de http://).

Informatia din cache manager este impartita in sectiuni - iata doar cateva exemple (pentru lista completa consultati anexa materialului):

- **menu** - afiseaza toate sectiunile disponibile
- **info** - afiseaza statistici generale de functionare a serverului
- **objects** - afiseaza obiectele memorate in cache impreuna cu starea lor curenta

Cache manager-ul poate fi accesat in doua moduri:

- prin intermediul unei aplicatii CGI numita *cachemgr.cgi* care, odata instalata corect pe un server web (plus configurarea aferenta in *squid*), permite vizualizarea unor pagini HTML cu toate informatiile oferite de cache manager
- prin intermediul utilitarului *squidclient* folosind URL-uri de forma **mgr:*sectiune***; *squidclient* va formula automat cererile folosind in URL-uri protocolul special de cache manager:

```
bash # squidclient mgr:menu
bash # squidclient mgr:info
bash # squidclient mgr:objects
```

Accesul la cache manager este interzis din oficiu; pentru a-l permite este necesara adaugarea directivelor potrivite in *squid.conf*:

```
# acces la cache manager numai de pe statia locala
acl manager url_regex -i ^cache_object://
acl localhost src 127.0.0.1
http_access allow manager localhost
http_access deny manager
```

8.5. Controlul accesului

8.5.1. Principii de functionare

Pentru a permite in mod controlat accesul clientilor la serviciile oferite, squid foloseste un sistem de control bazat pe doua categorii de directive:

- directive de tip ACL (Access Control List). Un ACL defineste un grup de cereri, caruia mai apoi i se pot aplica reguli de autorizare. Grupul se poate alcatui pe criterii complexe, pornind de la simpla adresa IP sursa a clientului si ajungand la autentificare sau regex-uri aplicate continutului cererii
- directive de autorizare – sunt cele care dicteaza, pe baza de ACL-uri, care cereri vor fi sau nu onorate de catre squid

8.5.2. ACL-uri

8.5.2.1. Principii si sintaxa generala

Un ACL defineste un grup de cereri sau raspunsuri HTTP cu caracteristici comune, in functii de criterii alese de catre administratorul de server, in scopul luarii de decizii de autorizare asupra acestor grupuri. ACL-urile se definesc cu ajutorul directivei **acl** care are urmatoarele sintaxe posibile:

```
acl nume tip argumente ...
acl nume tip "/cale/catre/fisier" ...
```

Tipul unui ACL indica criteriul dupa care sunt selectate cererile ce il formeaza. Exista numeroase tipuri de ACL-uri, corespunzatoare multelor criterii dupa care pot fi identificate cererile primite de la clienti. Le prezentam pe cele mai importante:

- **src,dst** – permit definirea acl-ului in functie de adesa IP sursa sau destinatie a cererii
- **srcdomain,dstdomain** – selectie in functie de numele DNS al clientului sau al serverului de origine
- **srcdom_regex, dstdom_regex** – similar cu srcdomain si dstdomain, dar permit criterii complexe prin utilizarea de regular expressions
- **port** – se refera la portul serverului de origine (cel la care se conecteaza squid). Putem selecta astfel cererile adresate anumitor porturi
- **method** – se refera la tipul de cerere HTTP (GET, POST, CONNECT etc)
- **proto** – selectia cererilor in functie de protocolul folosit (HTTP, HTTPS etc)
- **time** – selectia cererilor in functie de data/ora la care survin
- **proxy_auth** – folosit impreuna cu alte directive pentru a implementa accesul bazat pe autentificare
- **maxconn** – pentru a limita numarul de conexiuni provenite de la aceeasi adresa IP
- **url_regex** – selectie in functie de URL-ul ce face subiectul cererii
- **urlpath_regex** – similar, dar se aplica numai pentru ultima parte a URL-ului (fara protocol si hostname)
- **browser** – selectie in functie de continutul headerului HTTP User-Agent trimis de client
- ACL-uri externe – sunt acl-uri care selecteaza cererile folosindu-se de programe externe. Squid este configurat sa trimita informatiile necesare catre programul extern, iar acesta ii raporteaza daca cererea a fost selectata sau nu.

Atunci cand un ACL contine mai multe elemente (ex: multiple adrese IP sau subnet-uri), lista de elemente poate fi definita in mai multe moduri:

- in fisierul de configurare squid.conf, folosind prima sintaxa prezentata mai sus
  - folosind o singura directiva acl
  - folosind cate o directiva acl per element
- intr-un fisier separat, folosind cea de-a doua sintaxa de mai sus. Util atunci cand numarul de elemente este mare

Pentru ilustrare alegem exemplul unui ACL care specifica adresele IP ale statiilor de la departamentul de contabilitate al unei firme. Cele 3 statii au adresele 10.0.0.1, 10.0.0.24 si 10.0.0.217:

- specificare in squid.conf
  - folosind o singura directiva acl

```
acl calculatoare_contabilitate src 10.0.0.1 10.0.0.24 10.0.0.217
```

  - folosind cate o directiva per element:

```
# laura
acl calculatoare_contabilitate src 10.0.0.1
# victor
acl calculatoare_contabilitate src 10.0.0.24
# elena
acl calculatoare_contabilitate src 10.0.0.217
```

- specificare intr-un fisier separat:

```
acl calculatoare_contabilitate src "/etc/squid/computere_conta"
# ...iar fisierul computere_conta poate contine contine inclusiv comentarii:

# laura
10.0.0.1
# victor
10.0.0.24
# elena
10.0.0.217
```

Squid efectueaza automat SAU logic intre elementele unui ACL. In exemplele de mai sus, acl-ul in cauza va fi format din acele cereri a caror adresa IP sursa este 10.0.0.1 SAU 10.0.0.24 SAU 10.0.0.217, indiferent de forma in care este definit ACL-ul.

Atentie! Ordinea elementelor dintr-un ACL conteaza! Squid confrunta cererea cu elementele ACL-ului in ordinea in care au fost incluse, si de aceea elementele care se potrivesc cel mai des trebuie plasate la inceputul listei.

Exista un ACL creat din oficiu: cel numit **all** si care corespunde tuturor cererilor primite de squid.

Nota: *pentru diagnosticarea ACL-urilor se poate genera informatie de logging mai bogata legata de procesarea lor folosind in squid.conf directiva debug_options:*

```
debug_options ALL,1  28,2
```

8.5.2.2. Particularitati ale ACL-urilor care folosesc regex-uri

Exista numeroase tipuri de ACL care aplica regex-uri pentru a-si selecta cererile. Regex-ul este aplicat unuia dintre parametrii cererii (ex: nume DNS al serverului, referrer, URL etc). In utilizarea ACL-urilor de acest fel trebuie tinut cont de urmatoarele aspecte:

- implicit, o cerere va fi selectata de catre ACL daca parametrul ei confruntat cu regex-ul contine un subsir de formatul specificat. Spre exemplu, daca scriem un ACL care aplica regex-ul *index\.[a-z]{3}* URL-ului cererii, vor fi selectate atat cererile care solicita *index.htm*, *index.php*, *index.asp*, dar si cererile in care numele serverului de origine este *index.com* sau cererile care solicita fisierul *index.html*, deoarece aceste ultime exemple contin subsiruri de formatul dorit, chiar daca restul continutului lor difera. Pentru a impune ca regex-ul sa se aplice intregului parametru verificat este necesara utilizarea caracterelor ^ si $:

```
# cererile al caror URL path este /index. urmat de 3 caractere
acl indexuri urlpath_regex ^\/index\.[a-z]{3}$
```

- regex-ul este implicit case sensitive. Pentru a il face case insensitive, el trebuie precedat de optiunea -i

```
# cererile al caror nume DNS destinatie face parte din domeniul example.com, case insensitive
acl example dstdom_regex -i \.ExAmPle.com
```

8.5.2.3. Identificare dupa adresa IP si port

Aceasta sectiune cuprinde tipurile de ACL *src*, *dst* si *port*:

- ACL-ul de tip **src** pune conditii in functie de adresa sursa a cererii (in speta cea a clientului). Se pot specifica multiple adrese sursa, intervale de adrese sau subnet-uri
- ACL-ul de tip **dst** selecteaza cererile in functie de adresa serverului de origine, cu aceeasi sintaxa
- ACL-ul de tip **port** selecteaza cererile in functie de portul destinatie al cererii (cel pe care asculta serverul de origine)

Exemple:

```
acl retele_private src 10.0.0.0/8 127.0.0.0/8 172.16.0.0/16-172.16.0.0/16
acl https port 443
```

La ACL-urile de tip dst se poate folosi pe post de destinatie si un nume DNS, insa acesta *nu va fi confruntat cu numele DNS prezent in URL-ul cererilor*; el este transformat in adresa IP la pornirea squid:

```
acl infoacad dst infoacademy.net # se va referi de fapt la adresa IP 89.45.172.151
```

8.5.2.4. Identificare dupa numele DNS al serverului sau clientului

Aceste tipuri de ACL permit selectarea cererilor in functie de numele DNS al serverului de origine sau clientului, cu urmatoarea deosebire importanta intre ele:

- numele DNS al serverului este cel cu care este accesat serverul, asa cum a fost el scris in URL (si transmis mai departe prin intermediul headerului HTTP Host). Atunci cand cererea foloseste adresa numerica a serverului, squid va incerca sa obtina numele corespunzator prin rezolutie DNS inversa; daca aceasta esueaza, numele folosit va fi automat *none*.
- numele DNS al clientului va fi intotdeauna obtinut prin rezolutie DNS inversa (si deci in multe cazuri nu va exista)

Exista urmatoarele tipuri de acl care permit selectarea cererilor in functie de numele DNS al serverului de origine, respectiv al clientului:

- **dstdomain** – acl-ul poate primi ca parametru numele exact al serverului (ex: www.server.ro) sau o formula care desemneaza un intreg domeniu (ex: .server.ro)
- **dstdom_regex** – acl-ul primeste ca argument un regex ce se aplica numelui de domeniu. Putem astfel selecta intregi familii de domenii in baza unui pattern comun
- **srcdomain** – filtrare dupa numele DNS al clientului, cu aceeasi sintaxa ca si dstdomain
- **srcdom_regex** – analog cu dstdom_regex dar aplicat numelui DNS de client

```
# toate cererile care au ca destinatie www.server.ro
acl server_ro dstdomain www.server.ro

# toate cererile adresate unor servere al caror nume se termina in server.ro
acl domeniu_server_ro dstdomain .server.ro

# toate cererile adresate domeniilor care contin in numele lor cuvantul news
acl dstdom_regex ^.*news.*$
```

8.5.2.5. Identificare dupa momentul de timp al cererii

S	Sunday
M	Monday
T	Tuesday
W	Wednesday
H	Thursday
F	Friday
A	Saturday

Cererile pot fi incluse in acl-uri in functie de data si ora la care sunt receptionate de catre squid, conform ceasului de pe masina acestuia. Un exemplu de utilizare ar fi filtrarea accesului la internet pentru angajatii unei companii pe perioada orelor de munca si activarea sa numai in pauze.

In acest scop exista acl-ul de tip time, cu urmatoarea sintaxa:

```
acl numeACL time [zile] [ora1:minute1-ora2:minute2]
```

Fiecare zi are o litera corespondenta (vezi tabelul alaturat); pentru a specifica mai multe zile pur si simplu se concateneaza literele corespunzatoare.

Ora de inceput (ora1:minut1) trebuie sa fie mai mica decat ora de final (ora2:minut2).

```
# weekend-ul
acl weekend time AS

# zilele saptamanii, orele 8-16
acl work time MTWHF 08:00-16:00
```

8.5.2.6. Identificare dupa numar de conexiuni per IP

Squid permite limitarea numarului de conexiuni ce vin de la aceeasi adresa IP cu ajutorul tipului de ACL **maxconn**. Acesta selecteaza cererile care vin de la IP-uri avand mai multe conexiuni decat pragul specificat:

```
# cuprinde numai cererile venite de la adrese IP care au peste 3 conexiuni deschise deja cu squid
acl nume maxconn 3
```

Ulterior constituirii acl-ului, acestor cereri li se pot aplica reguli de interzicere a accesului.

In scrierea de astfel de acl-uri trebuie tinut cont de urmatoarele aspecte:

- o singura adresa IP poate avea in spatele sau un numar mare de clienti, gratie unui dispozitiv care efectueaza NAT
- multe browsere folosesc mai multe conexiuni simultane pentru a obtine continutul unei pagini, deoarece nu se pot depune mai multe cereri simultan pe aceeasi conexiune. In acest fel, toate resursele unei pagini se incarca in paralel (sursa HTML, poze, CSS-uri etc) si prezentarea paginii este mai placuta pentru utilizator

8.5.2.7. Identificare dupa componente ale URL-ului

Squid poate tine cont de orice componenta a URL-ului in selectarea cererilor ce fac subiectul unui ACL. Daca *dstdomain_regex* folosea numele DNS al serverului destinatie, prezentam in continuare alte doua tipuri de ACL care utilizeaza si restul URL-ului:

- **url_regex** – acest tip de ACL isi selecteaza cererile aplicand un regex intregului URL. **Atentie! URL-ul este cel complet, care include protocol, nume/IP server, URL path si eventual query string!**
- **urlpath_regex** – functioneaza prin analogie, insa aplica regex-ul numai URL path-ului (care incepe cu /)

```
# cererile HTTP care au in query string parametrul lang
acl querystring urlpath_regex ^/.*\?lang=

# cererile HTTP destinate statiilor al caror nume incepe cu www si care solicita fisiere
index.php, index.html sau index.asp
acl index url_regex ^http://www\..*index.(php|html|asp)$
```

8.5.2.8. Identificare dupa tipul de cerere HTTP

Tipul de ACL care selecteaza cererile in functie de tipul de cerere HTTP folosit este **method**. Squid recunoaste cererile HTTP uzuale (GET, POST, PUT, HEAD, DELETE, CONNECT, TRACE, OPTIONS) si pe cele mai rar folosite sau adaugate ulterior specificatiei initiale HTTP (ex: cererile WebDAV).

In plus fata de acestea, Squid adauga un tip de cerere propriu – PURGE – folosit pentru a solicita eliminarea din cache a unei resurse folosind utilitare administrative (ex: squidclient). Accesul la acest tip de cerere trebuie bine controlat, de aceea in general aceasta cerere face subiectul unui ACL care permite accesul numai de pe masina locala sau de pe un set restrans de alte masini:

```
acl masini_purge src 127.0.0.1
acl purge method PURGE
http_access allow purge masini_purge    # permitem utilizarea metodei PURGE numai de pe localhost
http_access deny purge                  # restul statiilor nu pot utiliza PURGE
```

Cererea de tip CONNECT trebuie tratata cu atentie sporita, deoarece prin intermediul ei clientul solicita proxy-ului sa se conecteze la un alt server in numele sau si sa joace rolul de intermediar fara a incerca sa inteleaga continutul mesajelor traficate. Acest tip de cerere a fost gandit initial pentru intermedierea conexiunilor SSL/TLS intre clienti si serverele de origine, insa, deoarece proxy-ul nu verifica in niciun fel continutul mesajelor, clientul ar putea tunela orice fel de informatie (nu neaparat HTTPS), creand astfel o problema de securitate. Din acest motiv cererile de tip CONNECT trebuie permise numai daca sunt destinate portului HTTPS, 443:

```
acl port_SSL port 443
acl connect method CONNECT
http_access deny connect !port_SSL    # cererea CONNECT nu poate fi adresata altui port decat 443
```

8.5.3. Reguli de control al accesului

Regulile de autorizare sunt cele care permit sau interzic accesul la diferite resurse pentru ACL-urile definite anterior. Squid dispune de o multitudine de directive care utilizeaza ACL-uri; le enumeram pe cele mai importante:

- **http_access** – cea mai importanta directiva de control al accesului. Stabileste care dintre cererile HTTP primite de la clienti vor fi onorate si care nu
- **http_reply_access** – analog, dar permite filtrarea raspunsurilor primite de la serverele de origine pe baza caracteristicilor acestora (ex: tip de continut, cod HTTP de raspuns etc)
- **no_cache** – permite specificarea granulara a obiectelor ce nu trebuie salvate in cache
- **redirector_access** – filtreaza cererile trimise catre un redirector extern. Implicit, daca a fost definit un redirector, toate cererile sunt trimise catre el
- **log_access** – permite filtrarea cererilor care sunt consemnate in access_log. Cererile care nu patrund in log nu intra nici in calculul diverselor statistici mentinute automat de server

Sintaxa generala a unei astfel de directive este:

```
directivaAcces  allow|deny  [!]ACL1  [!]ACL2 ….
```

Pentru ca o cerere/raspuns sa primeasca verdict favorabil, ea trebuie sa se incadreze in TOATE acl-urile specificate. Cu alte cuvinte, atunci cand o directiva de control al accesului contine mai multe ACL-uri, squid efectueaza automat SI logic intre ele. Oricare dintre ACL-uri poate fi precedat de semnul exclamarii, acesta avand rol de negare (asadar selectand numai cererile care NU se incadreaza in acel ACL):

```
# interzicerea cererilor venite din reteaua 10.0.0.0/24 in timpul orelor de program
acl retea_10 src 10.0.0.0/24
acl ore_program time MTWHF 8:00-16:00
http_access deny retea_10 ore_program

# permitem accesul pentru reteaua 192.168.0.0/24 in afara orelor de program
acl retea_192 src 192.168.0.0/24
http_access allow retea_192 !ore_program
```

Ordinea in care sunt plasate directivele de acces *de acelasi fel* in fisier conteaza, deoarece ele realizeaza un mini-firewall. Fiecare cerere sosita este confruntata pe rand cu fiecare regula de acces pana este cand intalnita una care se potriveste. Aceasta va da verdictul asupra cererii:

```
# acces neconditionat de pe localhost, doar intre 8-16 pentru 10.0.0.0/24, interzis pentru restul
acl localhost src 127.0.0.1
http_access  allow  localhost
http_access  allow  retea_10  ore_program
http_access  deny   all

# Localhost va avea accesul permis gratie primei reguli (si deci regula deny nu i se mai aplica)
# Statiile din reteaua 10.0.0.0/24 vor avea accesul permis de a doua regula
# Restul de statii vor cadea sub incidenta ultimei reguli, care le interzice accesul
```

Atentie! Daca niciuna dintre regulile de acces nu se potriveste, verdictul va fi inversul ultimei reguli din lista (ex: daca ultima regula este de tip allow, atunci cererea va primi deny). Spre exemplu, daca fisierul de configurare contine doar regulile de mai jos:

```
acl retea_10 sc 10.0.0.0/24
http_access allow retea_10
```

atunci orice statie care face parte din alta retea decat 10.0.0.0/24 va avea accesul interzis (fara a adauga o regula speciala in acest scop!), deoarece ultima directiva de acces are drept verdict *allow*.

8.6. Controlul accesului pe baza de autentificare

8.6.1. Mecanism

Squid poate autoriza cererile si in functie de username-ul furnizat la autentificare, cu ajutorul ACL-urilor de tip **proxy_auth**. Acestea nu pot functiona independent, ci au nevoie de minim un mecanism de autentificare definit in fisier. Parametrii diverselor mecanisme de autentificare posibile se stabilesc cu ajutorul directivei **auth_param**.

Atunci cand un client se conecteaza pentru prima data la un proxy care a fost configurat sa solicite autentificare, el va primi inapoi un raspuns cu urmatoarele caracteristici:

- codul HTTP de raspuns este 407 (Proxy Authentication Required)
- raspunsul include headerul HTTP Proxy-Authenticate, ce indica mecanismele de autentificare suportate de proxy si eventuale informatii suplimentare necesare autentificarii (in functie de mecanism)

Ca urmare, la cererile urmatoare clientul trebuie sa furnizeze informatii de autentificare utilizand headerul HTTP Proxy-Authorization.

Observatie: *autentificarea solicitata de un proxy difera de autentificarea HTTP in aceea ca este utilizata numai intre proxy si client; headerele HTTP care o implementeaza nu circula mai departe, catre serverele de origine. Prin contrast, headerul Authorization, folosit de catre client pentru a se autentifica HTTP, este transmis ca atare de catre proxy la serverul de origine.*

Atentie! Autentificarea nu poate fi folosita atunci cand squid functioneaza in modul proxy transparent, deoarece clientul traieste cu impresia ca dialogheaza direct cu serverul web de origine, care foloseste alte headere pentru a solicita autentificare (Authorization etc). Atunci cand squid este configurat in modul transparent, autentificarea este deazactivata.

Pentru a implementa autentificarea este necesara prezenta unor baze de date cu username-uri si parole (asa-numitele “back-end-uri” de autentificare). Squid nu interactioneaza direct cu aceste baze de date, ci prin intermediul unor programe ajutatoare pe care le numeste “helper”. Acestea primesc de la squid o singura linie ce contine username-ul si parola separate prin spatiu si produc ca output o singura linie ce contine fie “OK”, fie “ERR”. In acest fel squid este degrevat de a implementa direct interfata cu o multitudine de back-end-uri, iar a scrie un helper este suficient de simplu.

Squid suporta diferite mecanisme de autentificare. Un mecanism reprezinta modul concret in care clientul dialogheaza cu serverul pentru a face dovada cunoasterii corectei combinatii user-parola. Enumeram cateva:

- **basic** – in cazul acestui mecanism, clientul trimite ca atare username-ul si parola catre server. Mecanismul este considerat nesigur deoarece datele de autentificare pot fi capturate, permitand accesul atacatorului la serviciile serverului proxy
- **digest** – mecanismul nu mai presupune transmiterea in clar a parolei. Proxy-ul genereaza o secventa aleatoare si o transmite clientului; acesta calculeaza o valoare de raspuns folosind ca ingrediente secventa primita, username-ul, parola, tipul de cerere HTTP si URI-ul, si o transmite inapoi serverului. Implicit, functia folosita pentru calculul valorii de raspuns este una de hashing (MD5)
- **NTLM** – mecanism folosit pentru autentificarea in domenii Microsoft

Desi nesigur, mecanismul *basic* este unul dintre cele mai folosite si mai bogate in helperi. Iata cele mai importante programe helper pentru mecanismul basic:

- **getpwnam** – permite autentificarea din baze de date in format passwd (atentie! varianta veche, pre-shadow, in care parolele erau si ele memorate in passwd!)
- **PAM** – autentificare folosind sistemul PAM (Pluggable Authentication Modules), care reprezinta la randul sau o interfata catre o multitudine de mecanisme si back-end-uri de autentificare
- **SQL** – autentificare din baze de date SQL (inclusiv MySQL)
- **SMB** – autentificare folosind un server Windows
- **MSNT** – autentificare intr-un domeniu Microsoft
- **LDAP** – autentificare folosind un server LDAP

8.6.2. Definirea parametrilor mecanismului de autentificare

Fiecare mecanism de autentificare dispune de o serie de parametri. Fiecare parametru se configureaza utilizand directiva **auth_param**, cu sintaxa urmatoare:

```
auth_param mecanism numeParametru [setari]
```

Mecanismul este unul dintre cele suportate (basic, digest etc). Lista de setari depinde de parametrul ales. Prezentam in continuare principalii parametri pentru mecanismul basic:

- **program** *cale argumente* – specifica modalitatea de apelare a helperului (calea explicita catre el ***si argumentele care ii trebuie pasate***). La instalarea din pachete, helperii sunt plasati de obicei in */usr/lib/squid* sau */usr/lib/squid3*

  Nota: *corecta functionare a helperului poate fi verificata manual, apelandu-l cu aceeasi linie de comanda pe care o configuram ca valoare a parametrului program! Dupa pornire, helperul asteapta sa i se furnizeze username si parola; acestea vor fi introduse de la tastatura, pe aceeasi linie, separate printr-un spatiu. Helperul va replica OK sau ERR*

- **children** *nrProcese* – specifica numarul de procese de autentificare pe care squid le porneste automat la lansarea sa in executie. Daca numarul este prea mic in raport cu rata de sosire a cererilor de autentificare, acesta poate constitui un factor limitator al performantelor serverului
- **concurrency** *nrCereriSimultane* – specifica numarul de cereri de autentificare simultane pe care helperul le poate procesa. Valoarea din oficiu este 0 (nu se proceseaza cereri in paralel). Unii helperi suporta paralelismul, in sensul in care squid trimite mai multe cereri fara sa astepte rapunsul inainte de a trimite cererea urmatoare
- **realm** *domeniu* – stabileste numele zonei parolate, asa cum va fi el afisat clientului. Browserul va asocia in cache-ul sau informatiile de autentificare cu acest realm
- **credentialsttl** *interval* – stabileste cat de des va contacta squid helperul in scopul verificarii informatiilor de autentificare. Clientul trimite informatii de autentificare impreuna cu fiecare cerere; la prima cerere, squid contacteaza helper-ul si, in caz de autentificare corecta, pastreaza in memorie rezultatul, revalidandu-l numai la expirarea intervalului specificat. Valorile sunt exprimate sub forma unui numar si a unei unitati de masura (ex: *1 minute*, *1 hour* etc), valoarea din oficiu fiind de 2h.

8.6.3. Implementare

Prezentam in continuare modalitatea de implementare a autentificarii folosind mecanismul *basic* si back-end-ul MySQL. Pasii sunt urmatorii:

- configurare squid
  - configurarea parametrilor mecanismului de autentificare folosind **auth_param**
  - scrierea unuia sau mai multor ACL-uri de tip **proxy_auth** care selecteaza cererile in functie de username-ul furnizat de client la autentificare
  - scrierea uneia sau mai multot reguli de acces care implica ACL-ul definit anterior
- configurare MySQL
  - crearea unei baze de date si a unei tabele. Tabela trebuie sa contina minim 2 coloane – username si parola
  - popularea tabelei cu useri virtuali
  - crearea unui cont MySQL pe care squid il va folosi pentru accesarea tabelei de autentificare si stabilirea privilegiilor corecte (SELECT pe tabela cu useri)
- testare
  - testare helper, in mod independent
  - testare a intregului ansamblu

Helperul folosit pentru interfatarea cu servere de baze de date se numeste **squid_db_auth**. Acest executabil poate primi urmatoarele argumente:

- **--user** *username* – stabileste username-ul folosit de squid pentru a se conecta la serverul de baze de date
- **--password** *parola* – analog pentru parola
- **--dsn** *string* – stabileste tipul de DBMS si numele bazei de date (implicit *DBI:mysql:database=squid*)

- **--table** *tabela* – stabileste numele tabelei cu useri/parole. Tabela trebuie sa aiba cel putin doua coloane (user si parola) si o eventuala a treia coloana daca se utilizeaza si conditia suplimentara (vezi mai jos). Numele implicit al tabelei este *passwd*
- **--cond** *conditie* – reprezinta conditia suplimentara de autentificare a unui user, si care va fi adaugata sub forma unei clauze WHERE la interogarea pe care squid o formuleaza catre serverul de baze de date. Implicit aceasta conditie este "enabled=1", ceea ce presupune existenta unei coloane numite *enabled*. Pentru a anula aceasta restrictie se poate folosi pe post de conditie *1* sau sirul vid *""*
- **--usercol** *coloanaUsername* – stabileste numele coloanei de username care va fi folosit in formularea interogarii. Valoare implicita: *user*
- **--passwdcol** *coloanaParola* – analog pentru coloana de parola. Valoare implicita: *password*. Parolele pot fi criptate cu MD5 (varianta din oficiu) sau cleartext. In al doilea caz este necesara utilizarea optiunii --plaintext.
- **--plaintext** – indica faptul ca coloana de parola contine parolele in clar

```
# stabilirea parametrilor pentru mecanismul basic
auth_param basic program /usr/lib/squid_db_auth --user squid --password 3qu1d --table useri
                                        --usercol username --passwdcol parola --plaintext --cond 1
auth_param basic realm "Accesul permis doar pe baza de autentificare"
auth_param basic children 3
auth_param basic credentialsttl 1 hour

# acl care selecteaza toate cererile autentificate corect
acl autentificati proxy_auth REQUIRED

#...sau, pentru a selecta doar cererile care furnizeaza anumite username-uri:
# acl autentificati proxy_auth user1 user2 user3

# regulile de control al accesului
http_access deny !autentificati
```

Atentie! Definirea ACL-ului de tip *proxy_auth* trebuie facuta abia dupa ce a fost definit cel putin un mecanism de autentificare!

Atunci cand nu dispunem de mediu grafic, testarea corectei functionari se poate realiza si dintr-un terminal, folosind comanda *wget* dupa cum urmeaza:

- se creeaza o variabila de mediu care contine adresa proxy-ului:

```
bash $ export http_proxy='http://localhost:8080'
```

- se paseaza ca optiuni lui wget username-ul si parola necesare pentru autentificare:

```
wget --proxy-user=user1 --proxy-password=pass1 -O - http://www.infoacademy.net
```

8.7. Functionarea ca reverse proxy

Functionarea in modul reverse proxy introduce urmatoarele diferente fata de scenariul forward:

- clientii nu mai trebuie configurati special sa foloseasca serverul proxy drept intermediar
- clientii nu sunt constienti de faptul ca comunica cu un server proxy. Din punctul lor de vedere, dialogul este purtat chiar cu serverul de origine
- proxy-ul trebuie sa raspunda la cererile clientilor ca si cum ar fi serverul de origine

Activarea modului reverse proxy (numit si "accelerator") se realizeaza folosind optiuni suplimentare pasate directivei *http_port* dupa cum urmeaza:

- **accel** - activeaza modul reverse proxy. In acest mod de lucru, squid va accepta si cereri HTTP obisnuite (cele care contin doar URL Path); in mod normal, cererile adresate unui proxy de catre clientii sai contin URL-ul integral al resursei dorite
- **defaultsite**=*nume* - stabileste numele DNS implicit atunci cand clientul, in mod eronat, nu trimite header Host

In continuare trebuie configurat serverul de origine de pe care squid va obtine continutul livrat clientilor, folosind directiva **cache_peer**. Aceasta directiva are in squid utilizari multiple: ea este folosita atat pentru a desemna servere de origine in scenariul reverse proxy, cat si pentru a indica cache-uri "prietene" cu care formeaza o ierarhie de cache-uri ce comunica intre ele prin ICP (Internet Cache Protocol) sau HTCP (Hypertext Caching Protocol).

```
http_port 1234 accel defaultsite=www.example.com
cache_peer 5.6.7.8 parent 80 0 originserver no-query name=example
```

Semnificatia parametrilor directivei cache_peer de mai sus este urmatoarea:

- **5.6.7.8** - adresa serverului de origine
- **parent** - indica relatia ierarhica intre squid si acest partener de dialog
- **80** - portul HTTP pe care asculta partenerul (serverul de origine)
- **0** - portul ICP/HTCP pe care asculta cache-ul partener. 0 indica faptul ca partenerul nu suporta ICP/HTCP
- **originserver** - cererile vor fi adresate acestui partener in formatul necesar pentru un server web (vezi explicatia de la parametrul *accel* al directivei http_port)
- **no-query** - squid nu va incerca sa comunice cu acest partener de dialog folosind ICP/HTCP
- **name** - stabileste numele asociat acestui partener

Pentru a ne asigura ca squid nu va trimite catre serverul de origine cereri care nu sunt responsabilitatea lui, putem controla accesul folosind directiva **cache_peer_access**:

```
acl thesite dstdomain www.example.com
http_access allow thesite
cache_peer_access example allow our_sites      # folosim numele atribuit partenerului mai sus
cache_peer_access example deny all
```

Atunci cand un acelasi site este gazduit pe mai multe servere de origine si dorim sa distribuim cererile intre aceste servere (load balancing) putem lista mai multi parteneri squid:

```
cache_peer IP_server_1 parent 80 0 originserver no-query round-robin
cache_peer IP_server_2 parent 80 0 originserver no-query round-robin
...etc...
```

In cazul in care exista un singur server de origine, insa acesta gazduieste mai multe site-uri si distinge intre ele in functie de numele cu care este accesat ("name-based virtual hosting"), scenariul nu va mai functiona cu setarile de pana acum, deoarece squid inlocuieste automat valoarea headerului Host: primit de la client cu IP-ul serverului de origine. Pentru a pastra headerul original este necesara optiunea **vhost** in directiva http_port:

```
http_port 1234 accel vhost defaultsite=www.example.com
```

Nota: *modul de configurare prezentat este valabil incepand cu squid 2.6. Versiunile anterioare foloseau alte directive pentru configurarea in modul reverse proxy.*

8.8. BIBLIOGRAFIE

- Mecanisme de caching
  - http://www.mnot.net/cache_docs/
  - www.web-caching.com
  - Headere HTTP care controleaza caching-ul: http://palisade.plynt.com/issues/2008Jul/cache-control-attributes/
  - Specificatia HTTP:
    - headerul Cache-Control: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9.1
    - mecanismul de caching: http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13 – si mai ales 13.2, Expiration model, si 13.3 – Validation Model
  - Comparatie intre strategiile de inlocuire a obiectelor din cache: http://www.hpl.hp.com/techreports/1999/HPL-1999-69.html
- Configurare

- Squid 3.0 configuration manual (visolve): http://www.visolve.com/squid/squid30/contents.php
- Squid config manual: http://www.squid-cache.org/Versions/v3/3.1/cfgman/
- Exemple de configurare: http://wiki.squid-cache.org/ConfigExamples/
- Autentificare:
  - Autentificare in squid: http://wiki.squid-cache.org/Features/Authentication
  - Autentificare MySQL: http://wiki.squid-cache.org/ConfigExamples/Authenticate/Mysql
  - Manpage helper autentificare SQL: http://manpages.ubuntu.com/manpages/maverick/man8/squid3_db_auth.8.html
- Squid debug sections: http://wiki.squid-cache.org/KnowledgeBase/DebugSections
- Squid ca reverse proxy: http://wiki.squid-cache.org/SquidFaq/ReverseProxy

- Diagnostic si monitorizare:
  - http://etutorials.org/Server+Administration/Squid.+The+definitive+guide/Chapter+14.+Monitoring+Squid/14.2+The+Cache+Manager/
  - Cache Manager: http://wiki.squid-cache.org/Features/CacheManager#Cache_manager_Access_Control_in_squid.conf
- Carti:
  - Squid proxy 3.0: beginner's guide (http://www.amazon.com/Squid-Proxy-Server-3-0-Beginners/dp/1849513902/ref=sr_1_2?ie=UTF8&qid=1299231726&sr=8-2)
  - Web Content Caching and Distribution: http://www.amazon.com/Content-Caching-Distribution-Fred-Douglis/dp/9048166268/ref=sr_1_1?ie=UTF8&qid=1299232242&sr=8-1
  - Squid: The Definitive Guide (http://www.amazon.com/Squid-Definitive-Guide-Duane-Wessels/dp/0596001622) – atentie insa la schimbarile produse in configurare odata cu squid 2.7!

8.9. ANEXA 1 - Sectiuni disponibile in Cache Manager

Section name	Description
menu	Cache Manager Menu
pconn	Persistent Connection Utilization Histograms
mem	Memory Utilization
diskd	DISKD Stats
squidaio_counts	Async IO Function Counters
config	Current Squid Configuration
comm_epoll_incoming	comm_incoming() stats
ipcache	IP Cache Stats and Contents
fqdncache	FQDN Cache Stats and Contents
idns	Internal DNS Statistics
redirector	URL Redirector Stats
basicauthenticator	Basic User Authenticator Stats
external_acl	External ACL stats
http_headers	HTTP Header Statistics
info	General Runtime Information
service_times	Service Times (Percentiles)
filedescriptors	Process Filedescriptor Allocation
objects	All Cache Objects
vm_objects	In-Memory and In-Transit Objects
io	Server-side network read() size histograms
counters	Traffic and Resource Counters
peer_select	Peer Selection Algorithms
digest_stats	Cache Digest and ICP blob
5min	5 Minute Average of Counters
60min	60 Minute Average of Counters

utilization	Cache Utilization
histograms	Full Histogram Counts
active_requests	Client-side Active Requests
openfd_objects	Objects with Swapout files open
store_digest	Store Digest
store_log_tags	Histogram of store.log tags
storedir	Store Directory Stats
store_io	Store IO Interface Stats
store_check_cachable_stats	storeCheckCachable() Stats
refresh	Refresh Algorithm Statistics
delay	Delay Pool Levels
forward	Request Forwarding Statistics
cbdata	Callback Data Registry Contents
events	Event Queue
client_list	Cache Client List
asndb	AS Number Database
carp	CARP information
userhash	peer userhash information
sourcehash	peer sourcehash information
server_list	Peer Cache Statistics