Capítulo 4

Detalles técnicos sobre la implementación del proyecto

En esta sección veremos los detalles de implementación del proyecto. En cada apartado se incluirá el código y las modificaciones que se hayan realizado al código del sistema NoCat para poder interactuar con él. Para entender esta sección del proyecto es conveniente leerse el apéndice A.1 referente al funcionamiento de nocat y el capítulo 3 con una descripción global del funcionamiento.

4.1. Control de acceso

Para el control de acceso se usan los scripts conexion, salida, cn_min y cn_hour. Estos scripts usan funciones definidas en el módulo ControlNocat.pm.

4.1.1. Conexión

El código fuente del script conexion es muy sencillo:

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

#--------------------------------------------------------------------------
# Script:     conexion
# Version:    1.0
# Argumentos: login id_cola ubicacion
# Descripcion: Llama a la funcion conexion del modulo ControlNocat con
#              los mismos parametros para registrar la entrada en la red
```

```
#             en la tabla conexiones.
#-------------------------------------------------------------------------

use lib '/usr/local/nocat/lib/';
use ControlNocat;
use strict;

my $login=$ARGV[0];
my $id_cola=$ARGV[1];
my $ubicacion=$ARGV[2];

open (STDERR,">/dev/null");
&conexion($login,$id_cola,$ubicacion);
close (STDERR);
```

Este script se invoca desde la pasarela en el momento que un usuario es autenticado y la pasarela acaba de cambiar sus reglas del cortafuegos, esto se realiza en el método permit de la clase gateway.

Para invocar el script conexión nos tenemos que conectar por ssh desde la pasarela al servidor. Para realizar esta conexión se ha creado un usuario llamado nocat en el servidor de autenticación que es el propietario del script. El problema es el método para autenticar a la pasarela desde el servidor, no se puede usar el sistema de introducir una clave. Lo que se ha hecho es usar el método con clave pública en ssh. Se generan en la pasarela un par de claves RSA, la clave pública y la clave privada, para hacer eso ejecutamos:

ssh-keygen -t rsa

Cuando nos pide una clave para proteger la clave privada pulsamos enter dos veces y dejamos la clave privada sin proteger. Esto no presenta problemas ya que se supone que la pasarela está protegida, en caso de que alguien pueda llegar a la clave privada querrá decir que el sistema de protección ya ha fallado. El comando anterior crea el directorio .ssh dentro del directorio home del usuario que lo ejecuta, y dentro de ese directorio está la clave privada (id_rsa) y la clave pública (id_rsa.pub). En nuestro caso hemos creado un usuario nocat también en la pasarela, y será este usuario el que pueda conectarse al servidor sin necesidad de introducir ninguna clave. Nos tenemos que asegurar que sólo pueda leer ese usuario esas claves, para eso tecleamos:

chmod 700 /.ssh

Una vez hecho esto, creamos en el servidor de autenticación el usuario nocat y copiamos la clave pública generada antes en /home/nocat/.ssh/authorized_keys. En el fichero de configuración (/etc/ssh/sshd_config) debemos tener:

```
PubkeyAuthentication yes
```

Una vez que reiniciemos el servicio (/etc/init.d/sshd restart), el usuario nocat de la pasarela podrá conectarse por ssh al servidor sin necesidad de introducir ninguna clave.

En principio se invocó el script de conexión directamente desde el proceso gateway de la pasarela, pero se comprobó que se introducía mucho retardo mientras se establecía la conexión y se comprobaba la clave. La solución que se adoptó fue crear un proceso que se ejecuta en background en la pasarela (p_remoto) y atiende peticiones continuamente. Las peticiones tienen el siguiente formato:

servidor script [parámetros]

El servidor es la dirección del servidor, el script es el nombre del script que se ejecuta remotamente, y una serie de parámetros opcionales. El script lo busca en el directorio /usr/local/nocat/bin del servidor.

El modo de funcionamiento es escribir en un archivo FIFO una línea con el formato anterior y el proceso p_remoto se encarga de atender la petición y ejecutar el script deseado. El p_remoto crea un hijo cada vez que tiene que atender una petición, y se queda esperando a que termine.

De esta manera, cuando se produce una conexión se libera al proceso gateway de esperar la conexión ssh, permitiéndole continuar con su ejecución normalmente.

Pasemos a ver como quedó el método permit de la clase gateway. Introducimos unas líneas de comentarios para indicar el lugar donde se insertaron las líneas.

```
sub permit {
    my ( $self, $peer, $class ) = @_;
    my $fw = $self->firewall( GatewayAddr => $peer->gateway_ip );
    my $action;

    # Stash the peer object for future use.
    #
    $self->add_peer( $peer );

    # Update its expiration timestamp.
    #
    $peer->timestamp(1);

    # Get *our* notion of what the peer's service class should be.
    #
    $class = $self->classify( $peer, $class );

    my $prior_class = $peer->status;
```

```
    if ( $prior_class ne $class ) {
    # Insert the rule for the new class of service...
    #
    $fw->permit( $class, $peer->mac, $peer->ip );

    # *BEFORE* removing the rule for the *old* class of service, if any.
    # This way we don't drop packets for stateful connections in the
    # event of service upgrade.
    #
    if ( $prior_class and $prior_class ne DENY ) {
      $self->log( 5, "Upgrading ", $peer->user,
        " from $prior_class to $class service." );

      $fw->deny( $prior_class, $peer->mac, $peer->ip );
      $action = "Upgrade";
    } else {
      $self->log( 5, "User ", $peer->user, " permitted in class $class
               with rate ",$peer->rate,"Kbit and ceil ", $peer->ceil,"Kbit" );
      $action = PERMIT;

#  Codigo insertado para pasar a p_remoto una peticion que ejecute el script
#   conexion en el servidor y registre la conexion en la tabla conexiones
#
      my $servidor=$self->{AuthServiceAddr};
      my $login=$peer->user;
      my $ubicacion=$self->{Ubicacion};
      my $id=$peer->id_cola;
      open (FIFO,">/home/nocat/param_fifo");
      print FIFO "$servidor conexion $login $id $ubicacion";
      close FIFO;

#  Insertamos clase para la calidad de servicio
#
     $fw->ins_qos($peer->ip, $peer->id_cola, $peer->rate,
                                              $peer->ceil, $peer->prio);
    }
    $peer->status( $class );
#  Guardamos los parametros anteriores
#
    $peer->ceil_anterior( $peer->ceil);
    $peer->rate_anterior( $peer->rate);
  }
```

```
  elsif ( $peer->ceil_anterior  ne $peer->ceil or
            $peer->rate_anterior ne $peer->rate)
  {#Si cambian los parametros de conexion
    $self->log(5,"Upgrading ", $peer->user, "from Rate",
    $peer->rate_anterior,"Kbit Ceil", $peer->ceil_anterior,"Kbit to ",
       $peer->rate,"Kbit", $peer->ceil,"Kbit");
    $fw->del_qos($peer->ip, $peer->id_cola);
    $fw->ins_qos($peer->ip, $peer->id_cola, $peer->rate,
                                                $peer->ceil, $peer->prio);
    $peer->ceil_anterior( $peer->ceil);
    $peer->rate_anterior( $peer->rate);
    $action = "Renew";
  }
  else
  {
    $self->log( 5, "User ", $peer->user, " renewed in class $class
          with rate", $peer->rate,"Kbit and ceil", $peer->ceil,"Kbit" );
    $action = "Renew";
  }
  # Tell the parent process about it.
  $self->notify_parent( $action => $peer );
}
```

4.1.2. Salida

El proceso de salida es muy similar al de entrada en la red:

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

#--------------------------------------------------------------------------
# Script:     salida
# Version:    1.0
# Argumentos: login
# Descripcion: Llama a la funcion salida del modulo ControlNocat con
#              el login para registrar la salida de la red en la tabla
#              conexiones y en el historico.
#--------------------------------------------------------------------------

use lib '/usr/local/nocat/lib/';
use ControlNocat;
use strict;

open (STDERR,">/dev/null");
```

```
my $login=$ARGV[0];
&salida($login);
close(STDERR);
```

Al igual que en el caso de conexión, este script es invocado desde la pasarela usando el proceso p_remoto. El momento de invocar la salida es en el método deny de la clase gateway, justo después de cambiar las reglas del cortafuegos para que el usuario no pueda seguir navegando. Vamos a ver las líneas introducidas en este método:

```
sub deny {
    my ( $self, $peer ) = @_;
    my $mac = $peer->mac;

    # if we don't know the peer's MAC address, it must have been
    # an incidental connection (e.g. notification) that can be ignored.
    return unless $mac or $self->{IgnoreMAC};

    $peer = $self->remove_peer( $peer )
      or return $self->log( 4, "Denying unknown MAC address $mac?" );

    my $class = $peer->status;
# Liberamos el identificador de cola
#
    if (not $class or $class eq DENY)
    {
      my $id=$peer->id_cola;
      if ( $id and $id ne "")
      {
        my $servidor=$self->{AuthServiceAddr};
        open (FIFO,">/home/nocat/param_fifo");
        print FIFO "$servidor libera_id $id";
        close FIFO;
        $self->log(5,"Liberamos el identificador de cola $id");
      }
      return $self->log( 7, "Denying peer $mac without prior permit." );
    }

    $self->log( 5, "User ", ( $peer->user || $peer->ip ),
        " denied service. Connected since " ,
        scalar localtime $peer->connect_time, "." );

    my $fw = $self->firewall( GatewayAddr => $peer->gateway_ip );
    $fw->deny( $class, $mac, $peer->ip );
```

```
# Eliminamos la clase del QoS que corresponde a ese usuario
#
    $fw->del_qos( $peer->ip, $peer->id_cola );

# Codigo insertado para pasar a p_remoto una peticion que ejecute el
# script salida en el servidor y registre la salida en la tabla conexiones
#
    my $servidor=$self->{AuthServiceAddr};
    my $login=$peer->user;
    my $id=$peer->id_cola;
    open (FIFO,">/home/nocat/param_fifo");
    print FIFO "$servidor salida $login";
    close FIFO;

    $peer->status( DENY );

    # Tell the parent process about it.
    $self->notify_parent( DENY, $peer );
}
```

4.1.3. p_remoto

Este es el proceso que se ejecuta en background en la pasarela para atender las peticiones del proceso gateway. El código es el siguiente:

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

#--------------------------------------------------------------------
# Script:      p_remoto
# Version:     1.0
# Autor:       Jesús Martín Ruiz
# Argumentos:  Ninguno
# Descripcion: Crea un proceso en background para atender peticiones.
#              Las peticiones consisten en invocar un proceso en el
#              servidor por ssh. Recibe las peticiones a través de un
#              fichero fifo.
#--------------------------------------------------------------------

use strict;
use POSIX;
```

```
#Demonizamos
my $childpid = fork;
exit if $childpid;
setpgrp (0, $$);

#Controlamos la ejecucion del demonio
die "Ya se esta ejecutando" if (-e "/var/run/p_remoto.pid");
open(FILE,">/var/run/p_remoto.pid");
print FILE $$;
close(FILE);

#Recogemos las senales que devuelven los procesos hijos
$SIG{CHLD}="IGNORE";

#Archivo fifo para recoger peticiones
my $fifo = '/home/nocat/param_fifo';
unlink $fifo if (-e $fifo);
unlink $fifo if (-p $fifo);

if (system("mknod $fifo p "))
{
  die "No puede crear la fifo: $! ";
}

my $args;
while (1)
{
  #Abrimos la fifo
  open(FIFO,"<$fifo") or die "No se puede abrir: $!";

  #Leemos los datos
  $args = <FIFO>;
  close FIFO;

  #Creamos un hijo para atender la peticion
  my $pid = fork();
  if ($pid == 0)
  {

    (my $servidor,my $action,my @param)= split (/ /,$args);
    system("/usr/bin/ssh","nocat\@$servidor",
           "/usr/local/nocat/bin/$action @param");
```

```
    exit;
  }
}
```

4.1.4. Comprobación a cada minuto

Para la comprobación a cada minuto se ha creado el script cn_min, cuyo código es el siguiente:

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

#-------------------------------------------------------------------------
# Programa:  cn_min
# Version:   1.0
# Autor:     Jesus Martin Ruiz
# Descripcion: Crea un demonio que se ejecuta a cada minuto para
#            ejecutar la funcion check_1min, que compruba las conexiones
#            activas que han expirado y deben ser expulsados del sistema.
#-------------------------------------------------------------------------

use lib '/usr/local/nocat/lib/';
use ControlNocat;
use DBI;
use Getopt::Std;
use strict;

#Usuario de la base de datos
my $user = USER;
my $passwd = PASSWD;

#Espedifica manejador y base de datos
my $driver = DRIVER_DB;

#Conectamos con la base de datos
my $dbh = DBI->connect($driver, $user, $passwd)
  or &log ( "\nError ($DBI::err): $DBI::errstr\n");

my %opt;
getopts("F?" => \%opt);

if ( $opt{"?"} )
{
```

```
  die <<END;
Argumentos de Control Nocat a cada minuto:
   -F  Modo en foreground.
   -?  Esta ayuda.
END
}

if ( $opt{"F"} )
{
  print"Modo Foreground\n";
  &check_1min($dbh);
}
else
{
  my $childpid =fork;
  if ($childpid != 0)
  {
    #El padre acaba
    exit;
  }
  else
  {
    setpgrp (0, $$);
    die "Ya esta corriendo" if (-e "/var/run/cn_min.pid");
    open(FILE,">/var/run/cn_min.pid");
    print FILE $$;
    close(FILE);

    my @hora=localtime(time);

    #Esperamos a que los segundos esten en 59
    my $espera=59-$hora[0];
    sleep($espera);

    #Borramos el archivo de bloqueo por si se ha matado el demonio
    # cuando estaba dentro de la funcion.
    unlink("/home/nocat/check_1min.lock")
      if (-e "/home/nocat/check_1min.lock");

    while (1)
    {
      my @hora=localtime(time);
      #Esperamos a que los segundos esten en 00
```

```
      while ( $hora[0] != 0 )
      {
        @hora=localtime(time);
      }
      &check_1min($dbh);

      #Esperamos al siguiente minuto
      sleep(59);
    }
  }
}

#Cerramos la conexion
$dbh->disconnect;
```

Este script hay que ejecutarlo en el servidor al iniciar el sistema y se queda en background ejecutandose permanentemente. Cuando se produce cualquier incidencia guarda un registro en nocat.log en el home del usuario nocat.

4.1.5. Comprobación a cada hora

Para la comprobación a cada hora se ha creado el script cn_hour. Lo único que hace este script es conectarse con la base de datos y llamar a la función check_1hora del módulo ControlNocat.pm.

Para ejecutar este script se programa una tarea cron[1] en el servidor. Para programarla hacemos lo siguiente:

crontab -u nocat -e

Con esto editamos el crontab del usuario nocat, entonces añadimos una línea como la siguiente, para que ejecute el archivo cn_hour en el minuto 05 de cada hora.

05 * * * * /usr/local/nocat/bin/cn_hour

4.1.6. Comprobación de datos de activación

La comprobación de datos se realiza en el script cgi-bin/login. Se llama a la función check_datos la primera vez que se autentica. El lugar exacto lo mostramos a continuación:

[1]Para información del servicio cron veáse el apéndice A.2.

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

###
##
# login
#
# The CGI that does the deed.
#
#  * Present a form
#  * Check it when filled in
#  * Notify the connecting IP of the outcome
#  * Inform and optionally redirect the user
#
# RJF & SDE, 7.4.01
#
# License: GPL.
##
###

#use lib '/usr/local/nocat'; # or wherever.
use lib '../lib/';
use NoCat qw( ANONYMOUS );
use strict;
# El paquete que contiene check_datos y consulta_qos
use ControlNocat;

my $authserv = NoCat->auth_service( ConfigFile => $ENV{NOCAT} );
my $cgi = $authserv->cgi;
my $params = $cgi->Vars;

# Debug configuration setup.
$authserv->check_config(qw(
    LoginForm FatalForm RenewForm LoginOKForm ExpiredForm
    LoginGreeting LoginMissing LoginBadUser LoginBadPass
));

$authserv->log( 7, sprintf( "User %s from %s requests %s",
    $params->{user} || "UNKNOWN", $cgi->remote_host,
    lc( $params->{mode} ) || "form" )
);

# Figure out which image button was clicked
# (since they don't have value="" attributes).
if (my ($button) = grep { defined $params->{"mode_$_.x"} }
    qw( login skip logout ))
{
```

```
    delete $params->{$_} for ( "mode_$button.x", "mode_$button.y" );
    $params->{mode} = $button;
}

# Have we filled in the form yet?  No?  If not, present one.
$authserv->display( LoginForm => "LoginGreeting" ) unless $params->{mode};

# Verify prerequisites.
$authserv->display( FatalForm =>
   "Your MAC address is undefined.  Problem with the gateway?" )
      unless $params->{mac};
$authserv->display( FatalForm =>
   "Your gateway token is undefined.  Problem with the gateway?" )
      unless $params->{token};

# If the user skipped authentication...
if ( $params->{user} eq ANONYMOUS or $params->{mode} =~ /^skip/io )
{
    $params->{user} = ANONYMOUS;
    delete $params->{member};

# Otherwise, attempt to authenticate the user.
}
else
{
    # Are we just missing required fields?
    $authserv->display( LoginForm => "LoginMissing"   )
       unless $params->{user} and $params->{pass};

    # Does the login info match what we have on file?
    my $user = $authserv->user->fetch( $params->{user} );

    $authserv->display( LoginForm => "LoginBadUser" ) unless $user->id;
    $authserv->display( LoginForm => "LoginBadPass" )
         unless $user->authenticate( $params->{pass} );

    # Set the service class based on the user's authorization (if any).
    my $member = join( " ", $user->groups );
    $params->{member} = $member if $member;

# Guardamos el classid,ceil y prio en los parametros
#
    unless ($params->{id_cola})
    {
      my @qos=&consulta_qos($params->{user});
```

```
      $params->{id_cola}=$qos[0];
      $params->{rate}=$qos[1];
      $params->{ceil}=$qos[2];
      $params->{prio}=$qos[3];
    }
}

# Finally, notify the gateway (and the user) as to the outcome.
my ( $form, $gw );

# Either we're requesting the renewal popup box...
if ( $params->{mode} =~ /^popup/io )
{
    $form = ( $params->{gateway} ? "PassiveRenewForm" : "RenewForm" );
    $params->{redirect} = $authserv->renew_url;

# Or we're either logging in, or renewing, in which case, notify the gateway.
}
elsif ($gw = $authserv->notify( Permit => $params ))
{
  if ( $gw->{Error} )
  {
    # Oddly enough, this isn't really success.
    $form = "ExpiredForm";

  }
  elsif ( $params->{mode} =~ /^renew/io )
  {
    if ( $params->{gateway} )
    {
      $form = "PassiveRenewForm";
      # $params->{redirect} = $gw->{redirect};
    }
    else
    {
      $form = "RenewForm";
      $params->{redirect} = $authserv->renew_url( $gw );
    }
  }
  else
  {

# Muestra mensaje de error si la funcion check_datos no devuelve un 1
#
    $authserv->display(LoginForm=>"Error de autenticacion")
```

```
      unless (check_datos($params->{user},$params->{ubicacion}));

    $form = "LoginOKForm";
    # $params->{redirect} = $gw->{redirect} if $gw->{redirect};
    $params->{popup} = $authserv->popup_url( $gw );
    # set the javascript *link* to the popup box.
    # warn "+ redirect:[$params->{redirect}] popup:[$params->{popup}]\n";
  }

# Or something's really wrong.
}
else
{
    my $err = $!;
    if ($err =~ /Connection refused/io) {
      $authserv->display( LoginForm => "Can't connect to your gateway.
      If it's behind a NAT'ed firewall, it needs to run in Passive Mode." );
    } else {
      $authserv->display( LoginForm => "Authentication error for
                          connection $params->{token}: $!" );
    }
}

$params->{logout} = $authserv->logout_url; # Make a logout link.

# Execute on the plan and tell a compelling story to the user.
$authserv->success( $form => $params );

# Fin
```

El código introducido para la comprobación de datos es el trozo que aparece en segundo lugar. El fragmento que hay introducido antes lo veremos en el apartado de calidad de servicio.

La función check_datos está en el módulo ControlNocat.pm. Los parámetros que recibe esta función son user y ubicación, ambos son variables del cgi. En el caso de user es rellenado por el usuario al enviar el formulario. La ubicación desde donde se estaba conectando el usuario no estaba disponible en el servidor en el sistema NoCatAuth. Para pasar desde la pasarela al servidor este parámetro se han seguido dos pasos:

1. Definir una nueva directiva en el archivo de configuración de la pasarela:

   ```
   Ubicacion [MAC de la interfaz externa]
   ```

   El valor puede ser el que nosotros queramos que se registre en el servidor, en

nuestro caso hemos optado por la MAC de la interfaz externa de la pasarela. Hemos optado por este valor para tener la seguridad de que sea único.

2. Modificar el método capture_params de la clase Passive.pm:

```
sub capture_params {
    my ( $self, $peer, $request ) = @_;
    return {
      mac      => $peer->id,
      token    => $peer->token,
      redirect => $request,
      timeout  => $self->{LoginTimeout},
      gateway  => $peer->socket->sockhost . ":$self->{GatewayPort}",
#  Nuevo parámetro que pasamos al servidor de autenticacion en la URL
#
       ubicacion => $self->{Ubicacion}

    };
}
```

Con estos dos pasos el script login tiene disponible el parámetro ubicación y puede usarlo para llamar a la función check_datos.

4.2. Calidad de servicio

Para empezar con la calidad de servicio se invoca el script inicioQoS desde dentro del script initialize.fw que es ejecutado al iniciar la pasarela. El script inicioQoS es el siguiente:

```
#!/bin/bash

#Creamos la discipllina de cola de subida
tc qdisc del dev $ExternalDevice root
tc qdisc add dev $ExternalDevice root handle 1 htb default 10

#Creamos la disciplina de cola de bajada
tc qdisc del dev $InternalDevice root
tc qdisc add dev $InternalDevice root handle 1 htb default 10

#Creamos la clase raiz de subida
tc class add dev $ExternalDevice parent 1: classid 1:2 htb rate
  $MaxBandWidthUp burst 2k
```

```
#Creamos la clase por defecto de subida
tc class add dev $ExternalDevice parent 1:2 classid 1:3 htb rate
  $MinBandWidthUp ceil $MaxBandWidthUp burst 2k prio 6
tc qdisc add dev $ExternalDevice parent 1:3 handle 3 sfq perturb 10

#Creamos la clase raiz de bajada
tc class add dev $InternalDevice parent 1: classid 1:2 htb rate
  $MaxBandWidthDown burst 2k

#Creamos la clase por defecto de bajada
tc class add dev $InternalDevice parent 1:2 classid 1:3 htb rate
  $MinBandWidthDown ceil $MaxBandWidthDown burst 2k prio 6
tc qdisc add dev $InternalDevice parent 1:3 handle 3 sfq perturb 10

# Marcamos los paquetes segun el tipo de servicio
iptables -A OUTPUT -t mangle -p tcp --dport 22 -j TOS --set-tos Minimize-Delay
iptables -A OUTPUT -t mangle -p tcp --dport 80 -j TOS
  --set-tos Maximize-Throughput
iptables -A OUTPUT -t mangle -p tcp --dport 443 -j TOS
  --set-tos Maximize-Throughput

# End
```

Una vez que han sido iniciadas las clases, la pasarela espera que alguien se conecte. Cuando alguien intenta conectarse y es redirigido al servidor, es cuando se ejecuta el script login.cgi, y cuando se crea el mensaje que recibe la pasarela con los parámetros de la conexión.

Nosotros necesitamos incluir en este mensaje el identificador de cola y los valores de los parámetros rate, ceil y prio. Para ello hemos utilizado la función consulta_qos, que devuelve los parámetros del grupo al que pertenece el usuario. Conretamente esta función consulta la tabla *grupos* y devuelve una array con los siguientes parámetros: id_cola, rate, ceil y prio. El id_cola es un parámetro muy delicado, ya que hay que sacarlo de la pila y volver a dejarlo en ella cuando deje de estar en uso. El script login.cgi se ejecutará una vez cada 45 segundos por cada usuario, y cada vez que se ejecuta va a llamar a la función consulta_qos. Para devolver el parámetro id_cola lo primero que hace esta función es mirar si el usuario está en la tabla *conexiones*, si está en ella coge el parámetro de esta tabla. Si el usuario no está conectado es que está haciendo login y hay que asignarle un identificador nuevo de la pila. En este caso sacamos un identificador de la pila. Cuando el mensaje sea recibido por la pasarela se invocará el script conexion y automáticamente se rellenará la tabla *conexiones* con el id_cola.

Podemos ver en la página 93 el script login.cgi, y en el primer fragmento comentado está incluido la recogida de parámetros con la función consulta_qos y su posterior asignación a las variables del cgi para que sean incluidos en el mensaje.

Cuando la pasarela recibe este mensaje lo desencripta y recibe los parámetros: id_cola, rate, ceil y prio. Veamos las modificaciones realizadas en el método punch_ticket de la clase Captive.pm, que es el que se encarga de comprobar la veracidad del mensaje y guardar los valores recibidos:

```
sub punch_ticket {
    my ( $self, $msg, $id ) = @_;
    my %auth = $msg->parse;

    my $client = $self->{Peer}{$id} ;
    if (not $client )
    {
#  Si no tiene id se termina y hay que devolver el id_cola a la pila del serv.
#
      if ($auth{Id_cola})
      {
        my $id=$auth{Id_cola};
        my $servidor=$self->{AuthServiceAddr};
        open(FIFO,">/home/nocat/param_fifo");
        print FIFO "$servidor libera_id $id";
        close FIFO;
        $self->log(5,"Liberamos el identificador de cola $id");
      }
      return  $self->log( 2, "Unknown ID notify from $id!" );
    }

# Si hay algun error hay que liberar el id_cola
#
    if ($client->user and $client->user ne $auth{User})
    {
      if ($auth{Id_cola})
      {
        my $id=$auth{Id_cola};
        my $servidor=$self->{AuthServiceAddr};
        open(FIFO,">/home/nocat/param_fifo");
        print FIFO "$servidor libera_id $id";
        close FIFO;
        $self->log(5,"Liberamos el identificador de cola $id");
      }
      return $self->log( 2,"Bad user/id match
```

```
                    from $id: $auth{User} != " . $client->user );
    }
    if ($client->token ne $auth{Token})
    {
      if ($auth{Id_cola})
      {
        my $id=$auth{Id_cola};
        my $servidor=$self->{AuthServiceAddr};
        open(FIFO,">/home/nocat/param_fifo");
        print FIFO "$servidor libera_id $id";
        close FIFO;
        $self->log(5,"Liberamos el identificador de cola $id");
      }
      return $self->log( 2, "Bad token match
                  from $id: $auth{Token} != " . $client->token );
    }

    if (not $self->{IgnoreMAC} and $client->mac ne $auth{Mac})
    {
      if ($auth{Id_cola})
      {
        my $id=$auth{Id_cola};
        my $servidor=$self->{AuthServiceAddr};
        open(FIFO,">/home/nocat/param_fifo");
        print FIFO "$servidor libera_id $id";
        close FIFO;
        $self->log(5,"Liberamos el identificador de cola $id");
      }
      return $self->log( 2,"Bad MAC match
                    from $id: $auth{Mac} != "  . $client->mac );
    }

    # Identify the user and class.
    $client->user( $auth{User} );
    $client->groups( $auth{Member} );

# Asignamos id_cola,ceil y prio
#
    $client->id_cola( $auth{Id_cola} );
    $client->rate( $auth{Rate} );
    $client->ceil( $auth{Ceil} );
    $client->prio( $auth{Prio} );
```

```
    # Store the new token away for when the peer renews its login.
    $client->token(1);

    # Perform the requested action.
    if ( $auth{Action} eq PERMIT ) {
      $self->permit( $client );
    } elsif ( $auth{Action} eq DENY ) {
      $self->deny( $client );
    }

    $self->log( 9, "Available MACs: @{[ keys %{$self->{Peer}} ]}" );

    return \%auth;
}
```

En el último fragmento es donde se asignan estos parámetros, como se puede observar se usan nuevos métodos en la clase peer.pm:

```
package NoCat::Peer;

.........

sub id_cola {
    my ( $self, $id_cola ) = @_;
    $self->{Id_cola} = $id_cola if defined $id_cola;
    return $self->{Id_cola};
}

sub rate {
    my ( $self, $rate ) = @_;
    $self->{Rate} = $rate if defined $rate;
    return $self->{Rate};
}

sub ceil {
    my ( $self, $ceil ) = @_;
    $self->{Ceil} = $ceil if defined $ceil;
    return $self->{Ceil};
}

sub prio {
    my ( $self, $prio ) = @_;
    $self->{Prio} = $prio if defined $prio;
```

```
    return $self->{Prio};
}
```

Estos métodos, a su vez definen variables de la clase Peer que guardan un valor para los parámetros id_cola, rate, ceil y prio de cada cliente.

Ya tenemos los parámetros disponibles en la pasarela como unas variables más del cliente. Para crear la nueva clase que corresponde a un cliente con sus parámetros usamos el script qos.fw:

```
#!/bin/sh

#
# Note: your PATH is inherited from the gateway process
#

action=$1
ip=$2
id_cola=$3
rate=$4Kbit
ceil=$5Kbit
prio=$6


if [ -z "$action" -o -z "$ip" -o -z "$id_cola" ]; then
  echo
  echo Usage: $0 [add\|del] [IP] [Classid] [OPTIONS]
  echo add OPTIONS: [Ceil] [Prio]
  echo Example: $0 add 10.0.0.105 20 100 5
  echo        : $0 del 10.1.1.1 20
  exit 1
elif [ "$action" = "add" ]; then
  if [ -z "$rate" -o -z "$ceil" -o -z "$prio" ]; then
    echo
    echo Usage: $0 [add\|del] [IP] [Classid] [OPTIONS]
    echo add OPTIONS: [Rate] [Ceil] [Prio]
    echo Example: $0 add 10.0.0.105 2F 35 100 5
    echo        : $0 del 10.1.1.1 20
    exit 1
  fi
fi

#Se crea una clase, se le conecta con una disciplina de colas y un filtro
```

```
# que dirige los paquetes del cliente a esa clase.
if [ "$action" = "add" ]; then
  tc class add dev $InternalDevice parent 1:2 classid 1:$id_cola htb
    rate $rate ceil $ceil burst 2k prio $prio
  tc qdisc add dev $InternalDevice parent 1:$id_cola handle $id_cola
    sfq perturb 10
  tc filter add dev $InternalDevice parent 1:0 protocol ip prio 0x$id_cola
    u32 match ip dst $ip classid 1:$id_cola

#Para borrarlo se hace en orden inverso
elif [ "$action" = "del" ]; then
  tc filter del dev $InternalDevice parent 1:0 protocol ip prio 0x$id_cola
    u32 match ip dst $ip classid 1:$id_cola
  tc qdisc del dev $InternalDevice parent 1:$id_cola handle $id_cola
    sfq perturb 10
  tc class del dev $InternalDevice parent 1:2 classid 1:$id_cola
else
    echo "FATAL: Accion incorrecta:  $action!"
    exit 1
fi

#
# Fin
#
```

Para invocar el scrip qos.fw se han creado tres métodos nuevos en la clase Firewall:

```
package NoCat::Firewall;
......
@REQUIRED   = qw( ResetCmd PermitCmd DenyCmd GatewayMode
                  InsertQoSCmd DeleteQoSCmd);
......
my @Perform_Export = qw(
    InternalDevice ExternalDevice LocalNetwork AuthServiceAddr DNSAddr
    GatewayAddr GatewayPort IncludePorts ExcludePorts AllowedWebHosts
    MembersOnly RouteOnly IgnoreMAC
    Policy1 Policy2 Policy3 Policy4 Policy5 Policy6
    MarkPolicy1 MarkPolicy2 MarkPolicy3 MarkPolicy4 MarkPolicy5 MarkPolicy6
    MaxBandWidthDown MaxBandWidthUp MinBandWidthDown MinBandWidthUp
);
......
sub ins_qos
{
```

```
  my $self = shift;
  $self->actua( InsertQoS => @_ );
}
sub del_qos
{
  my $self = shift;
  $self->actua( DeleteQoS => @_ );
}
sub actua {
  my ( $self, $action, $ip, $id_cola, $rate, $ceil, $prio ) = @_;

  my $cmd = $self->format( $self->{"\u${action}Cmd"}, {
Id_cola => $id_cola,
Rate    => $rate,
Ceil    => $ceil,
        Prio    => $prio,
IP      => $ip,
  });

  local %ENV = %ENV;
  $ENV{$_} =(defined( $self->{$_}) ? $self->{$_} : "" ) for @Perform_Export;
  $ENV{PATH} =
  "$FindBin::Bin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin";
  system $cmd;
}
```

Para que esto funcione se han definido en el archivo de configuración de la pasarela las directivas:

```
# Ancho de banda minimo y maximo por defecto
MinBandWidthDown 35Kbit
MinBandWidthUp 23Kbit
MaxBandWidthDown 140Kbit
MaxBandWidthUp 95Kbit
```

Los métodos ins_qos y del_qos del firewall se invocan en el momento que el usuario entra en el sistema y cuando sale del mismo respectivamente:

- El momento de entrar en el sistema es en el método permit[2] de la clase gateway. Se invoca el método ins_qos justo detrás de pasar a p_remoto la petición para invocar el script conexión.

[2]Veáse la sección 4.1.1.

- La salida del sistema se produce en el método deny[3]. Desde el método deny se llama a del_qos justo despues de cambiar las reglas del firewall.

Recapitulando, ya hemos visto como pasar los parámetros de calidad del servidor, como crear la nueva clase para ese usuario y como borrarla. Lo que nos falta por ver es la forma de devolver al servidor el identificador de cola de dicha clase, para que lo introduzca otra vez en la pila. Para esto se ha creado un pequeño script en el servidor que llama a la función libera_id del módulo ControlNocat.pm. Lo único que hace esta función es insertar una nueva fila en la tabla *id_cola* con el identificador devuelto. Este script debe ser invocado por la pasarela en varios casos:

- Cuando un usuario abandona el sistema. Independientemente del motivo por el que abandona el sistema, siempre se llama al método deny (de la clase gateway), si este usuario estaba activo. Dentro del método deny[4] se invoca el proceso para liberar el identificador de clase.
- Cuando un usuario ha intentado conectarse pero se produce algún tipo de error. Esto ocurre cuando un cliente recibe un identificador pero la conexión falla, ya sea por que expire el temporizador, o porque la MAC sea incorrecta, o el token... En este caso no se llama el metodo deny ya que todavia no se había permitido el accceso a la red, en cambio el servidor de autenticación si que ha proporcionado a ese cliente un id_cola que debe ser devuelto a la pila. Estas llamadas se realizan todas desde el método punch_ticket[5] de la clase captive.

Para invocar el script libera_id la pasarela vuelve a usar el proceso p_remoto.

4.3. Políticas de acceso

Las nuevas directivas definidas en el archivo de configuración son las siguientes:

```
# Definicion de puertos permitidos para cada politica
#
#   Las marcas 1,2 y 3 para pertenecer a Owner,Member o Public
#   La marca 4 no te permite la entrada.
#   El resto a definir libremente.

MarkPolicy1 5
Policy1 80 443
```

[3]Veáse la sección 4.1.2.
[4]Véase página 88.
[5]Véase la página 100.

```
MarkPolicy2 6
Policy2 22 21 80 443

MarkPolicy3 2
#Policy3

MarkPolicy4 2
#Policy4

MarkPolicy5 2
#Policy5

MarkPolicy6 2
#Policy6
```

Con esta definición vemos que hay dos políticas nuevas con las marcas 5 y 6 que permiten el acceso a los puertos definidos en Policy1 y Policy2. Para poder usar estas directivas se ha cambiado en la clase firewall las directivas que se exportan:

```
my @Perform_Export = qw(
    InternalDevice ExternalDevice LocalNetwork AuthServiceAddr DNSAddr
    GatewayAddr GatewayPort IncludePorts ExcludePorts AllowedWebHosts
    MembersOnly RouteOnly IgnoreMAC
    Policy1 Policy2 Policy3 Policy4 Policy5 Policy6
    MarkPolicy1 MarkPolicy2 MarkPolicy3 MarkPolicy4 MarkPolicy5 MarkPolicy6
    MaxBandWidthDown MaxBandWidthUp MinBandWidthDown MinBandWidthUp
);
```

Al iniciar el firewall se ha cambiado el script initialize.fw para que tenga en cuenta las nuevas marcas definidas, y se creen reglas que sólo permitan el acceso a los puertos definidos:

```
#!/bin/sh
##
#
# initialize.fw: setup the default firewall rules
#
.......
# Flush all user-defined chains
#
.......
if [ "$MembersOnly" ]; then
```

```
  classes="1 2"
else
  classes="1 2 3 5 6"
fi

# Handle tagged traffic.
#
for iface in $InternalDevice; do
    for net in $LocalNetwork; do
      for fwmark in $classes; do
        # Only forward tagged traffic per class
         $fwd -i $iface -s $net -m mark --mark $fwmark -j ACCEPT
#         $fwd -o $iface -d $net -m mark --mark $fwmark -j ACCEPT

        # Masquerade permitted connections.
        $nat -o $ExternalDevice -s $net -m mark --mark $fwmark -j MASQUERADE
      done

      # Allow web traffic to the specified hosts, and don't capture
      # connections intended for them.
      #
      if [ "$AuthServiceAddr" -o "$AllowedWebHosts" ]; then
        for host in $AuthServiceAddr $AllowedWebHosts; do
          for port in 80 443; do
            $nat      -s $net -d $host -p tcp --dport $port -j MASQUERADE
            $redirect -s $net -d $host -p tcp --dport $port -j RETURN
            $fwd -s $net -d $host -p tcp --dport $port -j ACCEPT
            $fwd -d $net -s $host -p tcp --sport $port -j ACCEPT
          done
        done
      fi

      # Accept forward and back traffic to/from DNSAddr
      if [ "$DNSAddr" ]; then
        for dns in $DNSAddr; do
          $fwd -o $iface -d $net -s $dns -j ACCEPT

          for prot in tcp udp; do
            $fwd -i $iface -s $net -d $dns -p $prot --dport 53 -j ACCEPT
            $nat -p $prot -s $net -d $dns --dport 53 -j MASQUERADE

            # Force unauthenticated DNS traffic through this server.
            # Of course, only the first rule of this type will match.
            # But it's easier to leave them all in ATM.
            #
```

```
            # Commented out for now, it's got a syntax issue I can't
            # quite fathom: "iptables: Invalid argument"
            #   --Rob
            #
            #$nat -i $InternalDevice -m mark --mark 4 -p $prot \
            # --dport 53 -j DNAT --to $dns:53
          done
        done
      fi
    done

    # Set packets from internal devices to fw mark 4, or 'denied', by default.
    $mangle -i $iface -j MARK --set-mark 4
done
......


#
#  Definimos las politicas correspondientes que esten definidas
#
if [ "$Policy1" ]; then
  for iface in $InternalDevice; do
    for port in $Policy1; do
      $ports -p tcp -i $iface --dport $port -m mark --mark $MarkPolicy1
          -j ACCEPT
      $ports -p udp -i $iface --dport $port -m mark --mark $MarkPolicy1
          -j ACCEPT
    done

    # Siempre se permite el acceso al gateway o no se podra hacer logout
    $ports -p tcp -i $iface --dport $GatewayPort -j ACCEPT
    $ports -p udp -i $iface --dport $GatewayPort -j ACCEPT

    # Se deshabilitan el resto de puertos
    $ports -p tcp -i $iface -m mark --mark $MarkPolicy1 -j DROP
    $ports -p udp -i $iface -m mark --mark $MarkPolicy1 -j DROP
  done
fi
if [ "$Policy2" ]; then
  for iface in $InternalDevice; do
    for port in $Policy2; do
      $ports -p tcp -i $iface --dport $port -m mark --mark $MarkPolicy2
          -j ACCEPT
      $ports -p udp -i $iface --dport $port -m mark --mark $MarkPolicy2
```

```
          -j ACCEPT
    done

    # Siempre se permite el acceso al gateway o no se podra hacer logout
    $ports -p tcp -i $iface --dport $GatewayPort -j ACCEPT
    $ports -p udp -i $iface --dport $GatewayPort -j ACCEPT

    # Se deshabilitan el resto de puertos
    $ports -p tcp -i $iface -m mark --mark $MarkPolicy2 -j DROP
    $ports -p udp -i $iface -m mark --mark $MarkPolicy2 -j DROP
  done
fi
if [ "$Policy3" ]; then
  for iface in $InternalDevice; do
    for port in $Policy3; do
      $ports -p tcp -i $iface --dport $port -m mark --mark $MarkPolicy3
          -j ACCEPT
      $ports -p udp -i $iface --dport $port -m mark --mark $MarkPolicy3
          -j ACCEPT
    done

    # Siempre se permite el acceso al gateway o no se podra hacer logout
    $ports -p tcp -i $iface --dport $GatewayPort -j ACCEPT
    $ports -p udp -i $iface --dport $GatewayPort -j ACCEPT

    # Se deshabilitan el resto de puertos
    $ports -p tcp -i $iface -m mark --mark $MarkPolicy3 -j DROP
    $ports -p udp -i $iface -m mark --mark $MarkPolicy3 -j DROP
  done
fi
if [ "$Policy4" ]; then
  for iface in $InternalDevice; do
    for port in $Policy4; do
      $ports -p tcp -i $iface --dport $port -m mark --mark $MarkPolicy4
          -j ACCEPT
      $ports -p udp -i $iface --dport $port -m mark --mark $MarkPolicy4
          -j ACCEPT
    done

    # Siempre se permite el acceso al gateway o no se podra hacer logout
    $ports -p tcp -i $iface --dport $GatewayPort -j ACCEPT
    $ports -p udp -i $iface --dport $GatewayPort -j ACCEPT
```

```
    # Se deshabilitan el resto de puertos
    $ports -p tcp -i $iface -m mark --mark $MarkPolicy4 -j DROP
    $ports -p udp -i $iface -m mark --mark $MarkPolicy4 -j DROP
  done
fi
if [ "$Policy5" ]; then
  for iface in $InternalDevice; do
    for port in $Policy5; do
      $ports -p tcp -i $iface --dport $port -m mark --mark $MarkPolicy5
          -j ACCEPT
      $ports -p udp -i $iface --dport $port -m mark --mark $MarkPolicy5
          -j ACCEPT
    done

    # Siempre se permite el acceso al gateway o no se podra hacer logout
    $ports -p tcp -i $iface --dport $GatewayPort -j ACCEPT
    $ports -p udp -i $iface --dport $GatewayPort -j ACCEPT

    # Se deshabilitan el resto de puertos
    $ports -p tcp -i $iface -m mark --mark $MarkPolicy5 -j DROP
    $ports -p udp -i $iface -m mark --mark $MarkPolicy5 -j DROP
  done
fi
if [ "$Policy6" ]; then
  for iface in $InternalDevice; do
    for port in $Policy6; do
      $ports -p tcp -i $iface --dport $port -m mark --mark $MarkPolicy6
          -j ACCEPT
      $ports -p udp -i $iface --dport $port -m mark --mark $MarkPolicy6
          -j ACCEPT
    done

    # Siempre se permite el acceso al gateway o no se podra hacer logout
    $ports -p tcp -i $iface --dport $GatewayPort -j ACCEPT
    $ports -p udp -i $iface --dport $GatewayPort -j ACCEPT

    # Se deshabilitan el resto de puertos
    $ports -p tcp -i $iface -m mark --mark $MarkPolicy6 -j DROP
    $ports -p udp -i $iface -m mark --mark $MarkPolicy6 -j DROP
  done
fi
.......
# Llamamos a inicioQoS para que inicie las clases raiz y por defecto
```

```
#[ -x inicioQoS ] && inicioQoS
inicioQoS
.....
#
# Fin
#
```

Con esto se han creado una serie de reglas en el cortafuegos para impedir el acceso a unos puertos predeterminados a paquetes que estén marcados con una marca. ¿Cómo le decimos al firewall qué marca poner a los paquetes de un cliente determinado?

Recordemos el funcionamiento de NoCatAuth: la pasarela marca con 0x1, 0x2 y 0x3 a los paquetes de las clases Owner, Member y Public. En el método classify de la clase Gateway es donde clasifica a los usuarios. Si es uno de los usuarios contenidos en la directiva Owners de nocat.conf entonces pertenece a la clase OWNER. Si pertenece a un grupo de los que están en la directiva TrustedGroups entonces pertenecerá a la clase MEMBER (si en la directiva aparece la palabra Any cualquier miembro de un grupo estará en la clase MEMBER). El resto van a la clase PUBLIC. La pasarela recibe el grupo del servidor en el mensaje encriptado dentro del campo member. En ese campo el servidor ha enviado el campo network de la tabla *network* de la base de datos nocat.

Nosotros vamos a aprovechar este campo para enviar la marca con la que queremos que se marquen los paquetes de un usuario. En la pasarela pondremos dentro de TrutedGroups la palabra Any. De esta manera cualquier usuario sería marcado con MEMBER. Pues bien, hemos cambiado classify para que en lugar de marcar estos paquetes con MEMBER llame al método política que asigna la marca correspondiente. A continuación se muestran los cambios en classify:

```
sub classify {
    my ( $self, $peer ) = @_;
    my $user = $peer->user;
    my $class;

    if ($user and grep( $user eq $_, $self->owners ))
    {
      $class = OWNER;
    }
    else
    {
        $self->log(9, "User (@{[ $peer->groups ]}) v. trusted (@{[ $self->groups ]})" );
        my %prospect = map { $_ => 1 } $self->groups;
        if ( grep $_, @prospect{ $peer->groups } )
        {
             #$class = MEMBER;
             #En clase guardamos la politica a aplicar al usuario
```

```
            $class = "@{[$peer->politica]}";
        }
        else
        {
            $class = PUBLIC;
        }
    }
    return $peer->class( $class );
}
```

El método politica incluido en la clase Peer es:

```
sub politica {
  my ($self, $groups) = @_;
  my $list = $self->{Groups} ||= [];
  my $politica = "@$list";
  $politica =~ s/Any//;
  chop($politica);
  return "$politica";
}
```

En el método permit de la clase gateway se llama al método classify y se guarda en el parámetro class la marca que se debe aplicar a ese usuario. Ya tenemos en el método permit de la clase gateway la marca con la que debemos marcar los paquetes. Lo único que hay que hacer es pasar esa marca al método permit de la clase firewall, y este se encarga de pasarsela al script access.fw. En el script access.fw se han realizado una serie de modificaciones para que los paquetes se marquen con esa marca:

```
#!/bin/sh

##
# VERY simple access control script for leeenux
##

# Note: your PATH is inherited from the gateway process
#

action=$1
mac=$2
ip=$3
class=$4

if [ -z "$action" -o -z "$mac" -o -z "$ip" -o -z "$class" ]; then
```

```
    echo Usage: $0 [permit\|deny] [MAC] [IP] [Class]
    echo Example: $0 permit 00:02:2d:aa:bb:cc 10.0.0.105 member
    exit 1
fi

if [ "$action" = "permit" ]; then
    cmd=-A
elif [ "$action" = "deny" ]; then
    cmd=-D
else
    echo "FATAL: Bad action: $action!"
    exit 1
fi

if [ "$class" = "Owner" ]; then
    mark=1
elif [ "$class" = "Member" ]; then
    mark=2
elif [ "$class" = "Public" ]; then
    mark=3
#
# Marcamos los paquetes con la marca correspondiente
#
elif [ "$class" = "$MarkPolicy1" ]; then
    mark=$MarkPolicy1
elif [ "$class" = "$MarkPolicy2" ]; then
    mark=$MarkPolicy2
elif [ "$class" = "$MarkPolicy3" ]; then
    mark=$MarkPolicy3
elif [ "$class" = "$MarkPolicy4" ]; then
    mark=$MarkPolicy4
elif [ "$class" = "$MarkPolicy5" ]; then
    mark=$MarkPolicy5
elif [ "$class" = "$MarkPolicy6" ]; then
    mark=$MarkPolicy6
else
    echo "FATAL: Bad class: $class!"
    exit 1
fi

if [ "$IgnoreMAC" ]; then
    match_mac=""
else
```

```
    match_mac="-m mac --mac-source $mac"
fi

# Mark outbound traffic from this node.
iptables -t mangle $cmd NoCat $match_mac -s $ip -j MARK --set-mark $mark

# Mark inbound traffic to this node.
iptables -t filter $cmd NoCat_Inbound -d $ip -j ACCEPT

#
# Fin
#
```

4.4. Módulo ControlNocat.pm

En el módulo ConrolNocat.pm están todas las funciones que se han utilizado para el control de acceso.

4.5. Administración del sistema

Para la administración del sistema se han utilizado 11 scripts CGI escritos en perl. Además de los scripts se ha creado un módulo llamado AdminNocat que contiene las funciones que procesan los datos sacados de los scripts. En esta sección vamos a ver algunos de los detalles que cabe destacar.

Comprobación de datos

Para la comprobación de datos se han utilizado las expresiones regulares de Perl. Esta comprobación se ha realizado cuando se ha rellenado en un formulario un campo textfield o un campo password_field. Las expresiones utilizadas son las siguientes:

- Login y Cod_grupo. Para estos tipos sólo se han permitido caracteres alfanuméricos y el guión bajo:

  ```
  error unless ( $valor =~ /^\w+$/ );
  ```

- Nombre, apellidos y provincia. Se permiten los caracteres alfanuméricos, los espacios y las vocales acentuadas:

```
error if ( $valor =~ /([^\w\sñÑáÁéÉíÍóÓúÚ])/);
```

- Comentarios. Se permiten los mismos valores que para nombre y apellidos y además se permiten puntos y comas.

```
error if ( $valor =~ /([^\w\s\.,ñÑáÁéÉíÍóÓúÚ])/);
```

- Habitación, teléfono y parámetros de caudal. Sólo se permiten números:

```
error if ( $valor =~ /([^\d])/);
```

- DNI. En el campo DNI se permiten tres formatos: Letra seguido de números, números seguidos de letra y números solos:

```
error unless ( $valor =~ /^\d+[A-Za-z]*$/ or $valor =~ /^[A-Za-z]\d+$/)
```

- Fechas. Las fechas en español tienen el siguiente formato: DD-MM-AAAA HH:MM:SS. Como separadores se permiten los caracteres '.', ':', '-' y '#'. Para separar la fecha y la hora se permiten los mismos y además el espacio.

```
error unless ($fecha =~ /^(\d2)[-:.#](\d2)[-:.#](\d4)[-:.#\s]
                         (\d1,2)[-:.#](\d1,2)[-:.#](\d1,2)$/x)
```

Facturación

Para la parte de facturación se han tomado unas especificaciones por defecto que podrán ser cambiadas según las necesidades de cada empresa. Las premisas adoptadas son:

- Tarifa plana diaria. El campo tarifa de *activacion* contiene el precio por dia.
- Tarifa plana mensual. El campo tarifa de la tabla *activacion* tiene el precio por mes. Por defecto, en el momento de facturar se cuentan el número de meses que lleva el usuario, y además se suman los dias de más multiplicados por el precio de un mes dividido por 30.
- Pospago. En la tarifa pospago el campo de la tabla *activacion* tiene el precio por horas. Pero la facturación se realiza por segundos, es decir, si la conexión dura un segundo pues se cobra el precio por hora dividido por 3600.
- Bono. Los bonos cuestan el precio que cuesta en el campo tarifa.

Vamos a ver cómo se ha calculado el consumo y la facturación de los usuarios. Veremos un ejemplo donde las condiciones serán lo más restrictivas posibles. Vamos a ver el consumo del grupo GRUPO1 entre la fecha de inicio FECHA_INICIO y la fecha de fin FECHA_FIN. Para hallar el consumo se han utilizado las siguientes sentencias SQL:

1. Lo primero que se hace es crear una tabla temporal con la siguientes campos:

```
CREATE  TEMPORARY TABLE temp(
   login VARCHAR(8),
   tarifacion VARCHAR(10),
   inicio DATETIME,
   fin DATETIME,
   tarifa FLOAT);
```

2. Se genera la cláusula que se aplicará en la sentencia para seleccionar las filas implicadas en la facturación. Hay que tener en cuenta si se ha especificado un login, un grupo, o una ubicación concreta. En tal caso, hay que añadir a la cláusula parámetros tales como: *ubicacion='UBICACION1'* o *login='LOGIN1'*. En nuestro ejemplo se tienen dos cláusulas diferentes: una para el histórico de activación, y la otra para el histórico de conexiones. En la cláusula del histórico es suficiente añadir *grupo='GRUPO1'*. La cláusula del conexiones es un poco más complicada, ya que en la tabla *hco_conexiones* no está el campo grupo. Lo que se hace es un JOIN entre las tablas *usuarios* y hco_conexiones. La cláusula de conexiones empieza por *hco_conexiones.login = usuarios.login AND usuarios.grupo = 'GRUPO1'* .

3. Ya tenemos el grupo de usuarios que hay que seleccionar. Vamos a seguir delimitando la búsqueda, en primer lugar habrá que ver si estamos facturando o haciendo un informe de consumo. Si estamos facturando se añade a las cláusulas *AND facturado='no'*. De esta manera sólo se tnedrán en cuenta las filas que no estén facturadas previamente.

   La última restricción ha aplicar es la fecha. Se van a tomar todas las filas en las que se cumpla que: han empezado antes de la FECHA_FIN, y que han terminado después de FECHA_INICIO o todavía no han terminado.

4. Ya tenemos las dos sentencias que nos van a rellenar la tabla *temp* con todas las filas implicadas en la facturación:

```
INSERT INTO temp
  SELECT login,tarifacion,alta,baja,tarifa
    FROM hco_activacion
    WHERE (grupo = 'GRUPO1' AND
           alta < 'FECHA_FIN' AND
```

```
            (baja > 'FECHA_INICIO' OR baja IS NULL) AND
            tarifacion !='pospago');
INSERT INTO temp
  SELECT hco_conexiones.login,tarifacion,alta,baja,tarifa
    FROM hco_activacion,usuarios
    WHERE (hco_conexiones.login = usuarios.login AND
            usuarios.grupo = 'GRUPO1' AND
            inicio < 'FECHA_FIN') AND
            (fin > 'FECHA_INICIO' OR fin IS NULL) AND
            tarifacion ='pospago');
```

Con estas sentencia ya tenemos todas las filas implicadas en una misma tabla. En la figura 4.1 vemos los tres tipos de filas que puede contener la tabla según su fecha de inicio y su fecha de fin:

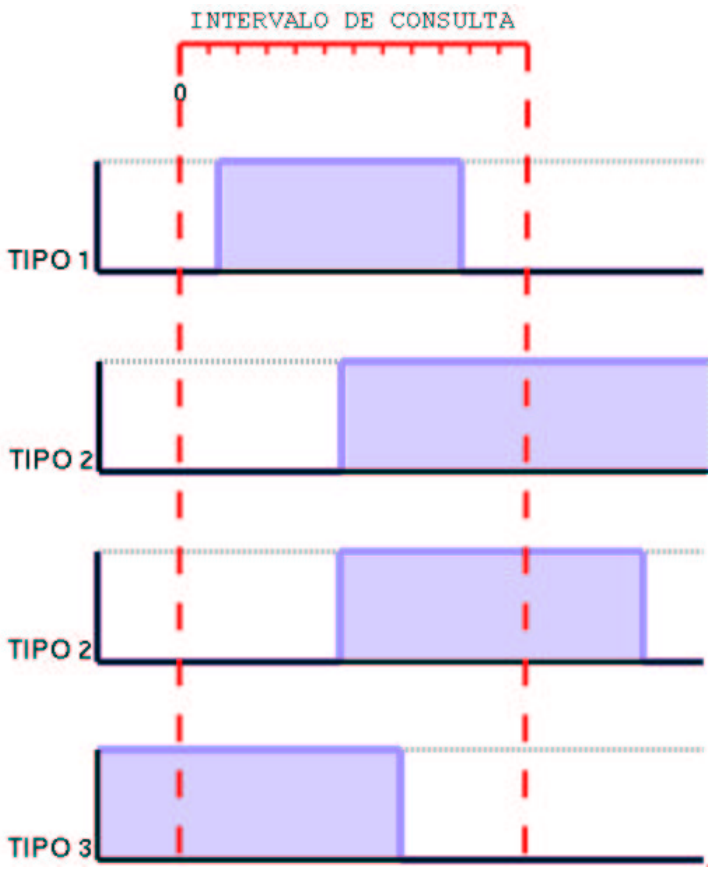

Figura 4.1: Tipos de filas dentro de la tabla *temp*.

A las filas de tipo uno no hay que hacerles nada. Se facturan tal cual están. A las filas de tipo dos se actualiza el tiempo de fin con el instante actual. Esto

se está realizando en una tabla temporal, así que no tendrá repercusión en el histórico.

```
UPDATE temp SET fin='FECHA_FIN'
                WHERE fin > 'FECHA_FIN' OR fin IS NULL;
```

Las filas del tipo tres sólo van a tener lugar cuando estemos realizando un informe de consumo, ya que en la facturación el intervalo de consulta tiene como inicio la fecha 0000-00-00. En los casos de informe las filas de tipo tres se cambiarán con la sentencia:

```
UPDATE temp SET inicio='FECHA_INICIO'
                WHERE inicio < 'FECHA_INICIO';
```

5. Después de hacer esto en la tabla temporal *temp* están todas las filas que hay que tarificar con la fecha de inicio, y la fecha de fin adecuadas. Es el momento de crear otra tabla temporal llamada *tabla_consumo* con el consumo de cada login:

```
CREATE TEMPORARY TABLE tabla_consumo(
  login VARCHAR(8),
  tarifacion VARCHAR (8),
  importe FLOAT);
```

6. Rellenamos en la tabla de consumo la filas de tarifa plana diaria:

```
INSERT INTO tabla_consumo
    SELECT login,tarifacion,
      (TO_DAYS(fin)-TO_DAYS(inicio))*tarifa
      FROM temp WHERE tarifacion='pdiaria';
```

7. Tarifa Plana Mensual:

```
INSERT INTO tabla_consumo
    SELECT login,tarifacion,
      PERIOD_DIFF(EXTRACT(YEAR_MONTH FROM fin),
                  EXTRACT(YEAR_MONTH FROM inicio))*tarifa +
      (TO_DAYS(fin)-
       TO_DAYS(DATE_ADD(inicio,INTERVAL
          PERIOD_DIFF(EXTRACT(YEAR_MONTH FROM fin),
                      EXTRACT(YEAR_MONTH FROM inicio)) MONTH)))* tarifa/30
      FROM temp WHERE tarifacion='pmensual';
```

8. Bono.

```
INSERT INTO tabla_consumo
    SELECT login,tarifacion,tarifa FROM temp
      WHERE tarifacion='bono';
```

9. Pospago.

```
INSERT INTO tabla_consumo
    SELECT login,tarifacion,
      ((TO_DAYS(fin)-TO_DAYS(inicio))*86400 +
        TIME_TO_SEC(EXTRACT(HOUR_SECOND FROM fin))-
        TIME_TO_SEC(EXTRACT(HOUR_SECOND FROM inicio))
      )*tarifa/3600
    FROM temp WHERE  tarifacion='pospago';
```

10. Como resultado de esto nos queda la tabla *tabla_consumo* con filas que contienen, el login, el tipo de tarifación y el importe. Para calcular el total de un usuario sólo tenemos que usar la sentencia:

```
SELECT sum(importe) FROM tabla_consumo
     WHERE login='LOGIN1';
```

Para saber el importe de cada tipo se basta con la sentencia:

```
SELECT sum(importe) FROM tabla_detalle
     WHERE login='LOGIN1' AND tarifacion='TARIFA';
```