```
#!/usr/bin/perl
################################################################
################
# Script permettant d'ecouter
# sur une interface reseau
# afin d'avoir les demandes DHCP en IPV6
#
# E.REUTER IFSTTAR Mars 2012
#
################################################################
################

use IO::Socket::INET6;
use IO::Socket::Multicast6;

use Net::DHCPv6::DUID::Parser;
use Proc::Daemon;
use Proc::PID::File;
$|=1;

use DBI();
use Net::PcapUtils;
use NetPacket::Ethernet qw( :strip );
use NetPacket::IP;# NetPacket::IP provides a set of routines
for assembling and disassembling packets using IP (Internet
Protocol)
use NetPacket::UDP; # NetPacket::UDP provides a set of
routines for assembling and disassembling packets using UDP
(User Datagram Protocol).

my $toPrint=0;
my $debug=1;

Proc::Daemon::Init();

# If already running, then exit
if (Proc::PID::File->running()) {
     print "all ready running !";
       exit(0);
}
close(STDOUT);
close(STDERR);

my $database="ocsweb";
my $hostname = "x.y.a.y.d";
my $user = "ocs";
my $pass = "ocs";

my $listenDev="eth3";

my $muticast=0;
my $s;
my $listenAddr="2001:660:xxxxx";
my $localLinkAddr='fe80::222:19ff:fe65:8076';
```

```
     my $dsn = "DBI:mysql:database=$database;host=$hostname";
        $dbh = DBI->connect($dsn, $user, $pass) or die "Echec
connexion";
if ($debug=0) {
     if ($multicast == 1) {
          # create a new IPv6 UDP socket ready to read
datagrams on port 1100
          my $s = IO::Socket::Multicast6->new(
                                    Domain=>AF_INET6,
                                    LocalPort=>547) or die
"Can't bind : $@\n";
           $s->mcast_add('FF02::1:2');
           $s->mcast_if($listenDev);

     } else {
     #
     # Ecoute en unicast


         $s = IO::Socket::INET6->new(listen    => 5,
                                LocalAddr => $listenAddr,
                                LocalPort => 547,
                        Domain => AF_INET6,
                                Proto     => 'udp') or die
"BBBBBBB No dhcpv6 daemon :$@ \n";

     }
}

my $start;
my $end;


 my($pcap_t) = create_pcap();

        unless ( $pcap_t ) {
            die "Unable to create pcap";
        }

        # Let's stop running as root.  Since we already
     # have our pcap descriptor, we can still use it.
        $EGID="$UNPRIV $UNPRIV";# setgid and setgroups()
        $GID=$UNPRIV;
        $UID=$UNPRIV; $EUID=$UNPRIV;

     # Capture packets forever.
        Net::Pcap::loop($pcap_t, -1, \&process_pkt, 0);

        # Technically, we shouldn't get here since the loop
        # is infinite (-1), but just in case, close and exit.
        Net::Pcap::close($pcap_t);


############################################################
##########
#
# Fonction de test du packet, pour savoir
```

```
# quelle est la demande du client V6
#
##############################################################
##########
sub testPacket {
  my $data=shift;


     print "code DHCP V6 ".ascii_to_hex(substr
($data,0,1))."\n";

     if (ascii_to_hex(substr($data,0,1)) eq "0d") {
          print "DHCP ipv6 renew \n";
           return 0;
          # Renew DHCP
     }

     if (ascii_to_hex(substr($data,0,1)) eq "0c") {
        if ($toPrint > 0) {
               print "dhcp ipv6 relay \n";
            }
            $start=52;
            $end=14;
    if (ascii_to_hex(substr($data,38,1)) eq "01") {
               print "\n DHCP V6 Solicit \n";
          }

          $start=42; # first Option
          if (ascii_to_hex(substr($data,$start,1)) eq "05") {
               print "\n DHCP V6 Renew \n";
               $start+=4;
          }

          if (ascii_to_hex(substr($data,$start,2)) eq "0008")
{
               $start+=6;
               print "\n Option Elapsed Time \n";
          }
          if (ascii_to_hex(substr($data,$start,2)) eq "0001")
{
               $start+=4;
               print "\n DHCP Solicit \n";
          }

     } else {
          # Normal dhcp ipv6 packet
          $start=14;
          $start=56;
          $end=14;
          #print "DHCP Direct \n";
     }


          if ($toPrint > 0) {
               print "data length ".length($data)." \n
\tStart = $start end =$end\n";
          }
          for (my $i=0;$i<length($data);$i=$i+1) {
```

```
                print ascii_to_hex(substr($data,$i,1)).":";

          }

          if ($toPrint > 0) {
                print "\n\n";
          }

          if ($toPrint > 0) {
                print ascii_to_hex(substr($data,$start,
$end))."\n";
          }
          my $tempEthernet=decodeV6(ascii_to_hex(substr
($data,$start,$end)));

          my $tempDuid;
          for (my $i=$start;$i<$start+$end;$i=$i+1) {
                $tempDuid.=ascii_to_hex(substr($data,
$i,1)).":";
                #print ascii_to_hex(substr($data,$i,1)).":";
          }
          chop($tempDuid);
          $tempEthernet=~ s/-/:/g;
          if ($toPrint > 0) {
                print "TEMP DUID $tempDuid \n Ethernet =
$tempEthernet\n";
          }
          updateDHCPv4($tempDuid,$tempEthernet);

          if ($tempEthernet eq '') { return 1;}
          return 0;


} # end testPacket


#####################
# Fermeture de la socket
$s->close();



sub decodeV6 {
     my $address=shift;
     my $p = new Net::DHCPv6::DUID::Parser;

  # Decode an example DUID
  $p->decode($address);

  # Print the type
  if ($toPrint > 0) {
       print "TYPE: ".$p->type(format => 'text')."\n";
  }

  ### prints 'TYPE: DUID-LL'

  if ($p->type == 1 || $p->type == 3) {
```

```
    # Format this like a MAC address if the link type was
Ethernet
    if ($p->iana_hw_type == 1) {
      if ($toPrint > 0) {
                print "MAC ADDRESS: ".$p->local_link_address
(format =>'ethernet_mac')."\n";
      }
      return $p->local_link_address(format =>'ethernet_mac');
    } else {
          if ($toPrint > 0) {
          print "LOCAL LINK ADDRESS: ".$p->
local_link_address."\n";
     }
    }

  }
  return;

}


sub decodeDHCPV6 {
     my $data=shift;

     my $length=$data[6];
     if ($toPrint > 0) {print "DHCP DUID Length : $length
\n"; }

#    decodeV6(
}


sub pr {
     my $chaine=shift;

     for (my $i=0;$i<length($chaine);$i=$i+1) {
          my $c=substr($chaine,$i,1);
               if ($c <= 9) {
                    $c=$c+30;
               } else {
               }
          }
}

################################################
#
# Fonction asscii to hex
#
################################################
sub ascii_to_hex ($)
    {
    (my $str = shift) =~ s/(.|\n)/sprintf("%02lx", ord
$1)/eg;
    return $str;
    }
    sub hex_to_ascii ($)
    {
    (my $str = shift) =~ s/([a-fA-F0-9]{2})/chr(hex $1)/eg;
```

```
    return $str;
    }


################################################
#
# Fonction pour MAj de la base de données du DHCP V4/V6
#
################################################
sub updateDHCPv4 {
     my $duid=shift;
     my $ether=shift;

     if ($ether eq '') { return; }
     my $query="update Historique set duid='$duid' where
Ether='$ether' \n";
     my $result = $dbh->prepare($query);
        $result->execute();
}

################################################
#
# Fonction recuperée sur Internet. Merci a son auteur...
#
################################################
sub create_pcap {
    my $promisc = 0;    # We're only looking for packets
destined to us,
                        # so no need for promiscuous mode.
    my $snaplen = 500; # Allows a max of 80 characters in the
domain name

    my $to_ms = 0;              # timeout
    my $opt=1;                          # Sure, optimisation
is good...
    my($err,$net,$mask,$dev,$filter_t);

    my $filter = "udp dst port 547 ";


     @devs = Net::Pcap::findalldevs(\%devinfo, \$err);
    for my $dev (@devs) {
        print "$dev : $devinfo{$dev}\n"
    }


    # Look up an appropriate device (eth0 usually)
    $dev = Net::Pcap::lookupdev(\$err);
    $dev or die "Net::Pcap::lookupdev failed.  Error was
$err";

    if ( (Net::Pcap::lookupnet($dev, \$net, \$mask, \$err) )
== -1 ) {
        die "Net::Pcap::lookupnet failed.  Error was $err";
    }

    $dev=$listenDev;
    # Actually open up our descriptor
```

```
    my $pcap_t = Net::Pcap::open_live($dev, $snaplen,
$promisc, $to_ms, \$err);
    $pcap_t || die "Can't create packet descriptor.  Error
was $err";

    if ( Net::Pcap::compile($pcap_t, \$filter_t, $filter,
$opt, $net) == -1 ) {
        die "Unable to compile filter string '$filter'\n";
    }


    # Make sure our sniffer only captures those bytes we want
in
    # our filter.
    Net::Pcap::setfilter($pcap_t, $filter_t);

    # Return our pcap descriptor
    $pcap_t;
}

# Routine to process the packet -- called by Net::Pcap::loop
()
# every time an appropriate packet is snagged.
sub process_pkt {
    my($user_data, $hdr, $pkt) = @_;

    my($src_ip) = 26;           # start of the source IP in
the packet
    my($dst_ip) = 30;           # start of the dest IP in the
packet
    my($domain_start) = 62;     # start of the domain in the
packet
    my($data);

    # extract the source IP addr into dotted quad form.
    my($source) = sprintf("%d.%d.%d.%d",
        ord( substr($pkt, $src_ip, 1) ),
        ord( substr($pkt, $src_ip+1, 1) ),
        ord( substr($pkt, $src_ip+2, 1) ),
        ord( substr($pkt, $src_ip+3, 1) ));

    # extract the destination IP addr into dotted quad form.
    my($destination) = sprintf("%d.%d.%d.%d",
        ord( substr($pkt, $dst_ip, 1) ),
        ord( substr($pkt, $dst_ip+1, 1) ),
        ord( substr($pkt, $dst_ip+2, 1) ),
        ord( substr($pkt, $dst_ip+3, 1) ));
     for (my $i=0;$i<length($pkt);$i=$i+1) {
                print ascii_to_hex(substr($pkt,$i,1)).":";

          }
    $data = substr($pkt, $domain_start);


     my $res=testPacket($data);
     if ($res > 0) {
          open (TMP,">>/tmp/logDHCPV6.log");
```

```
            for (my $i=0;$i<length($pkt);$i=$i+1) {
                 print TMP ascii_to_hex(substr($pkt,
$i,1)).":";

            }
            close(TMP);
      }

}
```