Scapy en 15 minutes

Guillaume Valadon / ANSSI & Pierre Lalet / CEA

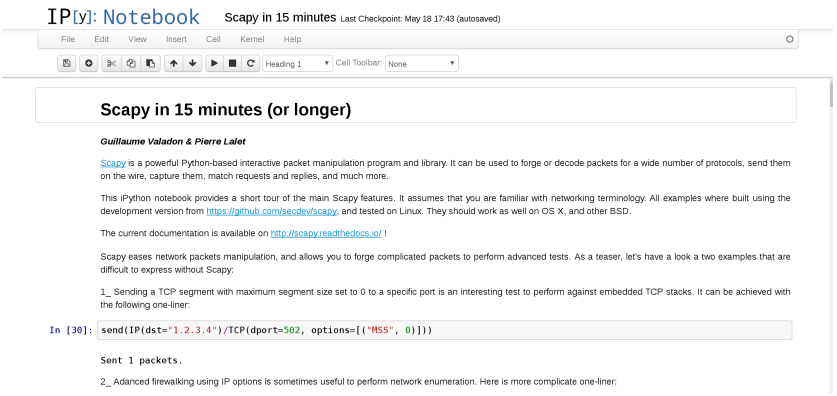
SSTIC - 3 juin 2016

Scapy ?

- manipulation de paquets en Python
  - développé par Philippe Biondi, dès 2003
  - maintenu par Guillaume & Pierre, depuis 2012
  - fonctionne nativement sous Linux
    - avec des modules sous Mac OS X, *BSD, & Windows
- une release stable à la fin de l'année
  - actuellement, 2.3.2

- disponible à https://github.com/secdev/scapy (https://github.com/secdev/scapy)
  - vous pouvez *starrer* le projet !
- documentation à http://scapy.readthedocs.io (http://scapy.readthedocs.io)

https://git.io/vrxMk (https://git.io/vrxMk)



Manipuler des paquets

- les paquets sont des objets
- l'opérateur / permet de superposer les paquets

In [2]:
```
packet = IP()/TCP()
Ether()/packet
```

Out[2]:
```
<Ether  type=IPv4 |<IP  frag=0 proto=tcp |<TCP  |>>>
```

- la fonction `ls()` permet de lister les champs des paquets

```
>>> ls(IP, verbose=True)
version    : BitField (4 bits)                = (4)
ihl        : BitField (4 bits)                = (None)
tos        : XByteField                       = (0)
len        : ShortField                       = (None)
id         : ShortField                       = (1)
flags      : FlagsField (3 bits)              = (0)
               MF, DF, evil
frag       : BitField (13 bits)               = (0)
ttl        : ByteField                        = (64)
proto      : ByteEnumField                    = (0)
chksum     : XShortField                      = (None)
src        : SourceIPField (Emph)             = (None)
dst        : DestIPField (Emph)               = (None)
options    : PacketListField                  = ([])
```

- Scapy choisit l'adresse IPv4 source, les adresses MAC, ...

In [7]:
```
p = Ether()/IP(dst="www.secdev.org")/TCP(flags="F")
p.summary()
```

Out[7]:
```
"Ether / IP / TCP 172.20.10.2:ftp_data > Net('www.secdev.org'):http F"
```

- chaque champ peut être facilement accédé

In [10]:
```
print p.dst      # première couche avec un champ dst, i.e. Ether
print p[IP].src  # accès explicite au champ dst de la couche IP

# sprintf() permet d'utiliser des 'formats strings' Scapy
print p.sprintf("%Ether.src% > %Ether.dst%\n%IP.src% > %IP.dst%")
```

```
3a:71:de:90:0b:64
172.20.10.2
b8:e8:56:45:8c:e6 > 3a:71:de:90:0b:64
172.20.10.2 > Net('www.secdev.org')
```

- un champ peut contenir plusieurs valeurs

In [11]:
```
[p for p in IP(ttl=(1,5))/ICMP()] # une suite de valeurs
```

Out[11]:
```
[<IP  frag=0 ttl=1 proto=icmp |<ICMP  |>>,
 <IP  frag=0 ttl=2 proto=icmp |<ICMP  |>>,
 <IP  frag=0 ttl=3 proto=icmp |<ICMP  |>>,
 <IP  frag=0 ttl=4 proto=icmp |<ICMP  |>>,
 <IP  frag=0 ttl=5 proto=icmp |<ICMP  |>>]
```

In [9]:
```
[p for p in IP()/TCP(dport=[22,80,443])] # des valeurs précises
```

Out[9]:
```
[<IP  frag=0 proto=tcp |<TCP  dport=ssh |>>,
 <IP  frag=0 proto=tcp |<TCP  dport=http |>>,
 <IP  frag=0 proto=tcp |<TCP  dport=https |>>]
```

Intérargir avec le réseau

- la fonction `sr1()` envoie un paquet et reçoit une réponse
- Scapy sait associer réponse et question

In [23]:
```
p = sr1(IP(dst="8.8.8.8")/UDP()/DNS(qd=DNSQR()))
p[DNS].an
```

```
Received 19 packets, got 1 answers, remaining 0 packets
Begin emission:
Finished to send 1 packets.
```

Out[23]:
```
<DNSRR  rrname='www.example.com.' type=A rclass=IN ttl=10011 rdata='93.184.216.34' |>
```

- la fonction `srp()` envoie une liste de trames, et retourne deux variables:
    1. `r` une liste de questions/réponses
    2. `u` une liste des paquets sans réponse

In [47]:
```
r, u = srp(Ether()/IP(dst="8.8.8.8", ttl=(5,10))/UDP()/DNS(ssqd=DNSQR(qname="www.exampl
e.com")))
r, u
```

```
Received 7 packets, got 6 answers, remaining 0 packets
Begin emission:
Finished to send 6 packets.
```

Out[47]:
```
 (<Results: TCP:0 UDP:0 ICMP:6 Other:0>,
  <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
```

In [48]:

```
# Accès au premier couple
print r[0][0].summary()  # paquet envoyé
print r[0][1].summary()  # paquet reçu

# Scapy a reçu un ICMP time-exceeded
r[0][1][ICMP]
```

```
Ether / IP / UDP / DNS Qry "www.example.com"
Ether / IP / ICMP / IPerror / UDPerror / DNS Qry "www.example.com."
```

Out[48]:

```
<ICMP  type=time-exceeded code=ttl-zero-during-transit chksum=0x50d6 reserved=0 length=0
 unused=None |<IPerror  version=4L ihl=5L tos=0x0 len=61 id=1 flags= frag=0L ttl=1 proto
=udp chksum=0xf389 src=172.20.10.2 dst=8.8.8.8 options=[] |<UDPerror  sport=domain dport
=domain len=41 chksum=0x593a |<DNS  id=0 qr=0L opcode=QUERY aa=0L tc=0L rd=1L ra=0L z=0L
 ad=0L cd=0L rcode=ok qdcount=1 ancount=0 nscount=0 arcount=0 qd=<DNSQR  qname='www.exam
ple.com.' qtype=A qclass=IN |> an=None ns=None ar=None |>>>>
```

- une liste de paquets peut être sauvée et lue depuis un fichier PCAP

In [50]:
```
wrpcap("scapy.pcap", r)

pcap_p = rdpcap("scapy.pcap")
pcap_p[0]
```

Out[50]:
```
<Ether  dst=f4:ce:46:a9:e0:4b src=34:95:db:04:3c:29 type=IPv4 |<IP version=4L ihl=5L tos
=0x0 len=61 id=1 flags= frag=0L ttl=5 proto=udp chksum=0xb6e3 src=192.168.46.20 dst=8.8.
8.8 options=[] |<UDP sport=domain dport=domain len=41 chksum=0xb609 |<DNS  id=0 qr=0L op
code=QUERY aa=0L tc=0L rd=1L ra=0L z=0L ad=0L cd=0L rcode=ok qdcount=1 ancount=0 nscount
=0 arcount=0 qd=<DNSQR  qname='www.example.com.' qtype=A qclass=IN |> an=None ns=None ar
=None |>>>>
```

- la méthode `command()` donne ce qu'il faut taper pour créér le même paquet

In [13]:
```
pcap_p[0].command()
```

Out[13]:
```
"Ether(src='34:95:db:04:3c:29', dst='f4:ce:46:a9:e0:4b', type=2048)/IP(frag=0L, src='19
2.168.46.20', proto=17, tos=0, dst='8.8.8.8', chksum=46819, len=61, options=[], version=
4L, flags=0L, ihl=5L, ttl=5, id=1)/UDP(dport=53, sport=53, len=41, chksum=46601)/DNS(aa=
0L, qr=0L, an=None, ad=0L, nscount=0, qdcount=1, ns=None, tc=0L, rd=1L, arcount=0, ar=No
ne, opcode=0L, ra=0L, cd=0L, z=0L, rcode=0L, id=0, ancount=0, qd=DNSQR(qclass=1, qtype=
1, qname='www.example.com.'))"
```

- la fonction `sniff()` permet de capturer des paquets

In [52]:
```
s = sniff(count=2)
s
```

Out[52]:
```
<Sniffed: TCP:0 UDP:2 ICMP:0 Other:0>
```

In [53]:
```
sniff(count=2, prn=lambda p: p.summary())
```

```
Ether / IP / TCP 172.20.10.2:52664 > 216.58.208.200:https A
Ether / IP / TCP 216.58.208.200:https > 172.20.10.2:52664 A
```

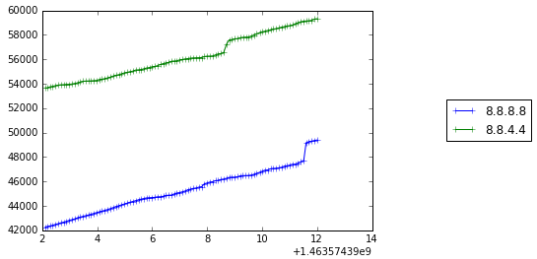
Out[53]:
```
<Sniffed: TCP:2 UDP:0 ICMP:0 Other:0>
```

Visualisation

- avec `srloop()`, on envoie 100 paquets vers 8.8.8.8 et 8.8.4.4
- la méthode `multiplot()` est utilisée pour afficher les valeurs des champs IP ID

In [25]:
```
ans, unans = srloop(IP(dst=["8.8.8.8", "8.8.4.4"])/ICMP(), inter=.1, timeout=.1, count=1
00, verbose=False)
```

In [26]:
```
%matplotlib inline
ans.multiplot(lambda (x, y): (y[IP].src, (y.time, y[IP].id)), plot_xy=True)
```



Out[26]:
```
[[<matplotlib.lines.Line2D at 0x7f2c2e113d10>],
 [<matplotlib.lines.Line2D at 0x7f2c2e113f10>]]
```

- le constructeur `str()` permet de construire un paquet tel qu'il sera envoyé sur le réseau

In [8]:
```
pkt = IP() / UDP() / DNS(qd=DNSQR())
repr(str(pkt))
```

```
'E\x00\x00=\x00\x01\x00\x00@\x11|\xad\x7f\x00\x00\x01\x7f\x00\x00\x01\x005\x005\x00)\xb6
\xd3\x00\x00\x01\x00\x00\x01\x00\x00\x00\x00\x00\x03www\x07example\x03com\x00\x00\x0
1\x00\x01'
```

Cette représentation étant compliquée, Scapy permet:

- de faire un "hexdump" de son contenu

In [18]:
```
hexdump(pkt)
```

```
0000   45 00 00 3D 00 01 00 00  40 11 7C AD 7F 00 00 01   E..=....@.|.....
0010   7F 00 00 01 00 35 00 35  00 29 B6 D3 00 00 01 00   .....5.5.)......
0020   00 01 00 00 00 00 00 00  03 77 77 77 07 65 78 61   .........www.exa
0030   6D 70 6C 65 03 63 6F 6D  00 00 01 00 01           mple.com.....
```

- de dumper le paquet couche par couche en affichant les valeurs des champs

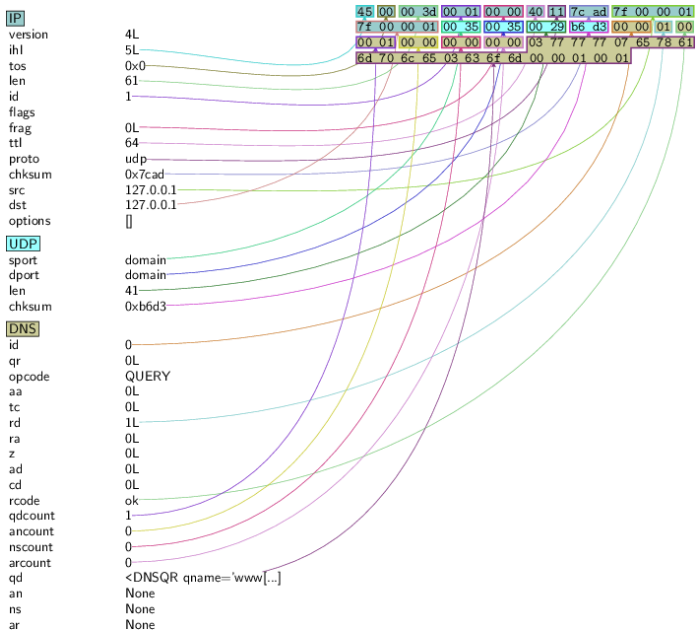
In [11]:
```
pkt.show()
```

```
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = udp
  chksum    = None
  src       = 127.0.0.1
  dst       = 127.0.0.1
  \options   \
###[ UDP ]###
     sport     = domain
     dport     = domain
     len       = None
     chksum    = None
###[ DNS ]###
        id        = 0
        qr        = 0
        opcode    = QUERY
        aa        = 0
        tc        = 0
        rd        = 1
        ra        = 0
        z         = 0
        ad        = 0
        cd        = 0
        rcode     = ok
        qdcount   = 1
        ancount   = 0
        nscount   = 0
        arcount   = 0
        \qd        \
         |###[ DNS Question Record ]###
         |  qname     = 'www.example.com'
         |  qtype     = A
         |  qclass    = IN
        an        = None
        ns        = None
        ar        = None
```

- d'afficher une jolie représentation du contenu du paquet

In [15]:
```
pkt.canvas_dump()
```

Out[15]:



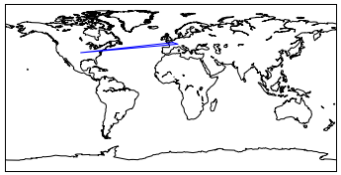
La fonction `traceroute()` appelle `sr(IP(ttl=(1,15))` et crée un objet `TracerouteResult`

In [22]:
```
ans, unans = traceroute('www.secdev.org', maxttl=15)
```

```
Begin emission:
Finished to send 15 packets.

Received 17 packets, got 15 answers, remaining 0 packets
   217.25.178.5:tcp80
1  192.168.46.254  11
2  172.28.0.1      11
3  80.10.115.251   11
4  10.123.205.82   11
5  193.252.98.161  11
6  193.252.137.74  11
7  193.251.132.183 11
8  130.117.49.41   11
9  154.25.7.150    11
10 154.25.7.150    11
11 149.6.166.166   11
12 149.6.166.166   11
13 217.25.178.5    SA
14 217.25.178.5    SA
15 217.25.178.5    SA
```

Les résultats peuvent être affichés de différentes façons, comme avec la méthode `.world_trace()` qui nécessite le module GeoIP de MaxMind (https://www.maxmind.com/)

In [23]:
```
ans.world_trace()
```



Out[23]:
```
[[<matplotlib.lines.Line2D at 0x7f2c23b62850>]]
```

La méthode `make_table()` permet une représentation textuelle compacte, par exemple pour un *port scanner* rudimentaire

In [29]:
```
targets = ["scanme.nmap.org", "nmap.org"]

ans = sr(IP(dst=targets)/TCP(dport=[22, 80, 443, 31337]), timeout=3, verbose=False)[0]

ans.extend(sr(IP(dst=targets)/UDP()/DNS(qd=DNSQR()), timeout=3, verbose=False)[0])

ans.make_table(lambda (x, y): (x[IP].dst,
                               x.sprintf('%IP.proto%/{TCP:%r,TCP.dport%}{UDP:%r,UDP.dport%}'),
                               y.sprintf('{TCP:%TCP.flags%}{ICMP:%ICMP.type%}')))
```

```
          45.33.32.156 45.33.49.119
tcp/22    SA           SA
tcp/31337 SA           RA
tcp/443   RA           SA
tcp/80    SA           SA
udp/53    dest-unreach -
```

Scapy en tant que module Python

Scapy peut facilement être utilisé pour faire ses propres outils comme, `ping6.py`:

In [ ]:

```
from scapy.all import *
import argparse

parser = argparse.ArgumentParser(description="A simple ping6")
parser.add_argument("ipv6_address", help="An IPv6 address")
args = parser.parse_args()

print sr1(IPv6(dst=args.ipv6_address)/ICMPv6EchoRequest(), verbose=0).summary()
```

Implémenter un nouveau protocole

L'ajout de nouveaux protocoles implique de créer un objet:

- qui hérite de `Packet`
- définit les champs dans la variables `fields_desc`

In [119]:
```
class DNSTCP(Packet):
    name = "DNS over TCP"

    fields_desc = [ FieldLenField("len", None, fmt="!H", length_of="dns"),
                    PacketLenField("dns", 0, DNS, length_from=lambda p: p.len)]

    # Cette méthode indique à Scapy que le payload doit être décodé avec DNSTCP
    def guess_payload_class(self, payload):
        return DNSTCP
```

Ce nouvel objet peut être directement utilisé pour construire et décoder un paquet

In [120]:
```
DNSTCP(str(DNSTCP(dns=DNS())))
```

Out[120]:
```
<DNSTCP  len=12 dns=<DNS  id=0 qr=0L opcode=QUERY aa=0L tc=0L rd=0L ra=0L z=0L ad=0L cd=
0L rcode=ok qdcount=0 ancount=0 nscount=0 arcount=0 |> |>
```

L'objet `StreamSocket` permet à Scapy d'utiliser une socket TCP

In [122]:
```
import socket

sck = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # création socket TCP
sck.connect(("8.8.8.8", 53))  # connection à 8.8.8.8 sur 53/TCP

# Creation d'une StreamSocket et définition de la classe par défaut
ssck = StreamSocket(sck)
ssck.basecls = DNSTCP

# Envoi de la requête DNS
ssck.sr1(DNSTCP(dns=DNS(qd=DNSQR(qname="www.example.com"))))
```

```
Received 1 packets, got 1 answers, remaining 0 packets
Begin emission:
Finished to send 1 packets.
```

Out[122]:
```
<DNSTCP  len=49 dns=<DNS  id=0 qr=1L opcode=QUERY aa=0L tc=0L rd=1L ra=1L z=0L ad=0L cd=
 0L rcode=ok qdcount=1 ancount=1 nscount=0 arcount=0 qd=<DNSQR  qname='www.example.com.'
  qtype=A qclass=IN |> an=<DNSRR  rrname='www.example.com.' type=A rclass=IN ttl=12101 rd
 ata='93.184.216.34' |> ns=None ar=None |> |>
```

Répondeurs

Scapy permet d'attendre une requête et d'envoyer une réponse, via l'objet `AnsweringMachine`

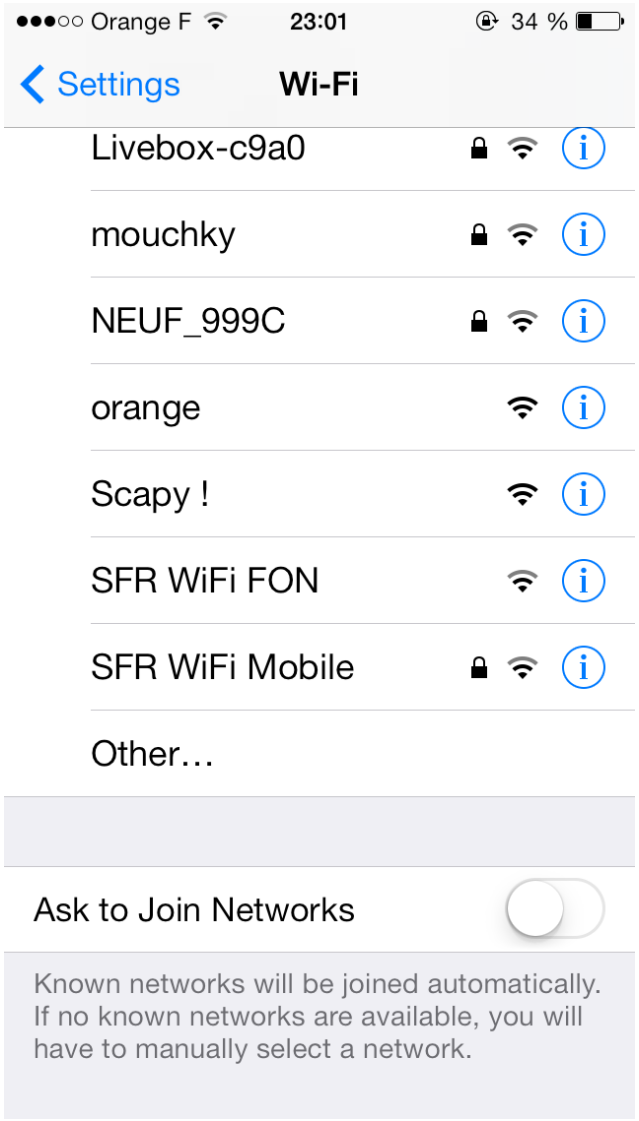
Deux méthodes sont nécessaires:

1. `is_request()` : retourne `True` si le paquet est la requête attendue
2. `make_reply()` : retourne le paquet qui peut ête envoyé

Note: dans l'exemple suivant l'interface Wi-Fi doit être mise en mode *monitor*.

In [ ]:

```
# Specifier l'interface Wi-Fi à utiliser
conf.iface = "mon0"

# Création du répondeur
class ProbeRequest_am(AnsweringMachine):
  function_name = "pram"

  mac = "00:11:22:33:44:55"

  def is_request(self, pkt):
    return Dot11ProbeReq in pkt

  def make_reply(self, req):

    rep = RadioTap()
    # Note: en fonction de la carte Wi-Fi, un autre entête que RadioTap() peut être néce
ssaire
    rep /= Dot11(addr1=req.addr2, addr2=self.mac, addr3=self.mac, ID=RandShort(), SC=Ran
dShort())
    rep /= Dot11ProbeResp(cap="ESS", timestamp=time.time())
    rep /= Dot11Elt(ID="SSID",info="Scapy !")
    rep /= Dot11Elt(ID="Rates",info='\x82\x84\x0b\x16\x96')
    rep /= Dot11Elt(ID="DSset",info=chr(10))

    return rep

# Démarrer le répondeur
#ProbeRequest_am()()  # décommenter pour tester
```


●●●○○ Orange F
23:01
34 %
Settings
Wi-Fi
Livebox-c9a0
mouchky
NEUF_999C
orange
Scapy !
SFR WiFi FON
SFR WiFi Mobile
Other…
Ask to Join Networks
Known networks will be joined automatically. If no known networks are available, you will have to manually select a network.

Automates

Scapy fournit une abstraction efficace pour construire des automates.

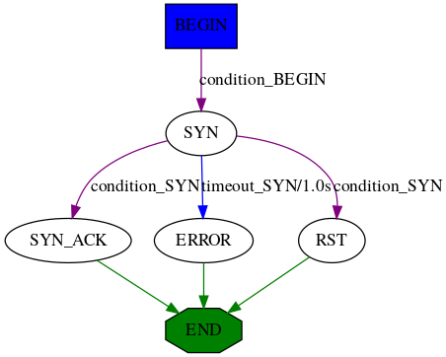
Il suffit d'hériter de l'objet `Automaton`, et de définir des méthodes spécifiques:

- states: elles utilisent le décorateur `@ATMT.state`. Habituellement, elles ne comportent pas de code utile
- conditions: elles utilisent les décorateurs `@ATMT.condition` et `@ATMT.receive_condition`. Elles définissent comment passer d'un état à un autre
- actions: elles utilisent le décorateur `ATMT.action`. Elles décrivent quoi faire, comment envoyer un paquet, lorsque l'on change d'état

In [ ]:

```
class TCPScanner(Automaton):

    # Définitions des états
    @ATMT.state(initial=1)
    def BEGIN(self):
        pass

    @ATMT.state()
    def SYN(self):
        print "-> SYN"

# [..]

    # Définitions des conditions
    @ATMT.condition(SYN)
    def condition_SYN(self):

        if random.randint(0, 1):
            raise self.SYN_ACK()
        else:
            raise self.RST()

    @ATMT.timeout(SYN, 1)
    def timeout_SYN(self):
        raise self.ERROR()
```

- la méthode `run()` démarre l'automate
- la méthode `graph()` affiche les relations entre états



Aller plus loin

- les **pipes** permettent des manipulations avancées: transfert entre interfaces, modifications à la volée, ...
- le module **nfqueue** de Python permet de maquetter des MitM rapidement

Ce qui va arriver prochainement !

- le support des certificats X.509 a été modifié par Maxence TURY
- Scapy peut désormais signer des certificats

In [ ]:
```
# Note: le PR ajoutant cette fonctionnalité n'est pas encore mergé
f = open("cert.der")  # Lecture d'un certificat
c = X509_Cert(f.read())

c.tbsCertificate.serialNumber = 0x4B1D  # Modification d'une valeur

k = PrivKey("key.pem")  # Nouvelle signature
new_x509_cert = k.resignCert(c)
```

- les OS dérivés de BSD (OS X, FreeBSD, OpenBSD, et NetBSD) vont être supportés nativement, sans modules externes.

```
# Note: le PR ajoutant cette fonctionnalité n'est pas encore mergé

git clone https://github.com/guedou/scapy-bpf

sudo ./run_scapy
>>> sys.platform

'darwin'

>>> conf.L3socket

<L3bpfSocket: read/write packets using BPF>
```

Questions ?

Issues ?

Pull requests ?