Introduction to protocol fuzzing using Scapy

Artjom Vassiljev
LuSec Briefings <lusec.al1us.net>
2009

From the Scapy homepage:

“Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. It can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery (it can replace hping, 85% of nmap, arpspoof, arp-sk, arping, tcpdump, tethereal, p0f, etc.). It also performs very well at a lot of other specific tasks that most other tools can't handle, like sending invalid frames, injecting your own 802.11 frames, combining technics (VLAN hopping+ARP cache poisoning, VOIP decoding on WEP encrypted channel, ...), etc.”

[*] Download page: http://www.secdev.org/projects/scapy/
[*] Latest version: 2.0.1-dev
[*] Supported platforms: Linux, *BSD, Windows

[+] Written in Python
[+] Easy to extend
[+] Import as a Python module and write your own tools
[+] Scapy uses the python interpreter as a command board, which means we can create loops, functions and assign variables right in Scapy

To run:

```
$ sudo sh run_scapy
Welcome to Scapy (2.0.1-dev)
>>> lsc()
    arpcachepoison : Poison target's cache with (your
                MAC,victim's IP) couple
    arping         : Send ARP who-has requests to
                determine which hosts are up
    bind_layers    : Bind 2 layers on some specific
                fields' values
    corrupt_bits   : Flip a given percentage or number
                of bits from a string
    […snip…]

>>> help(arping)
    arping(net, timeout=2, cache=0, verbose=None, **kargs)
        Send ARP who-has requests to determine which hosts are up
    arping(net, [cache=0,] [iface=conf.iface] [verbose=conf.verb])
-> None
        Set cache=True if you want arping to modify internal ARP-
Cache
```

Getting packet structure:

```
>>> ls()
…
IP:      IP
…
>>> ls(IP)
version    : BitField             = (4)
ihl        : BitField             = (None)
tos        : XByteField           = (0)
len        : ShortField           = (None)
id         : ShortField           = (1)
flags      : FlagsField           = (0)
frag       : BitField             = (0)
ttl        : ByteField            = (64)
proto      : ByteEnumField        = (0)
chksum     : XShortField          = (None)
src        : Emph                 = (None)
dst        : Emph                 = ('127.0.0.1')
options    : PacketListField      = ([])
```

Creating basic packets:

```
>>> a = IP(dst ="www.ee")/ICMP()
>>> a.show()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= icmp
  chksum= 0x0
  src= 92.254.191.73
  dst= Net('www.ee')
  \options\
###[ ICMP ]###
     type= echo-request
     code= 0
     chksum= 0x0
     id= 0x0
     seq= 0x0
```

Modify the packet layers:

```
>>> a.dst
Net('www.ee')
>>> a.dst = "www.se"
>>> a.dst
Net('www.se')
>>> a.payload.code
0
>>> a.payload.code = 1
>>> a.payload.code
1
```

```
Send the packet:

    sr     : Send and receive packets at layer 3
    sr1    : Send packets at layer 3 and return only the first answer
    srbt       : send and receive using a bluetooth socket
    srbt1  : send and receive 1 packet using a bluetooth socket
    srflood: Flood and receive packets at layer 3
    srloop : Send a packet at layer 3 in loop and print the answer each time
    srp    : Send and receive packets at layer 2
    srp1   : Send and receive packets at layer 2 and return only the first
answer
    srpflood   : Flood and receive packets at layer 2
    srploop: Send a packet at layer 2 in loop and print the answer each time
```

Send the packet #2:

```
>>> ans,unans = sr(a)
Begin emission:
..Finished to send 1 packets.
*
Received 3 packets, got 1 answers, remaining 0 packets
>>> ans
<Results: TCP:0 UDP:0 ICMP:1 Other:0>

>>> ans.hexdump()
0000 21:05:55.773467 IP / ICMP 92.254.191.73 > 194.204.33.19 echo-request 1 ==>
IP / ICMP 194.204.33.19 > 92.254.191.73 echo-reply 1 / Padding

0000   45 00 00 1C 97 28 00 00  37 01 EC 91 C2 CC 21 13 E....(..7.....!.
0010   5C FE BF 49 00 01 FF FE  00 00 00 00 00 00 00 00 \..I............
0020   00 00 00 00 00 00 00 00  00 00 00 00 00 00           ..............
```

Protocol fuzzing:

```
>>> a = fuzz(IP())
>>> a.show()
###[ IP ]###
  version= <RandNum>
  ihl= None
  tos= 208
  len= None
  id= <RandShort>
  flags= MF
  frag= <RandNum>
  ttl= <RandByte>
  proto= <RandByte>
  chksum= 0x0
  src= 92.254.191.73
  dst= 137.191.206.245
  \options\
```

Protocol fuzzing #2:

```
>>> a = fuzz(IP(dst="80.90.10.1"))
>>> a.show()
###[ IP ]###
  version= <RandNum>
  ihl= None
  tos= 208
  len= None
  id= <RandShort>
  flags= MF
  frag= <RandNum>
  ttl= <RandByte>
  proto= <RandByte>
  chksum= 0x0
  src= 92.254.191.73
  dst= 80.90.10.1
  \options\
```

Protocol fuzzing #3:

```
>>> send(a, count = 10, inter = 1, iface = "eth0")
..........
Sent 10 packets.
```