Use Scapy to build your own tools

You can use Scapy to make your own automated tools. You can also extend Scapy without having to edit its source file.

If you have built some interesting tools, please contribute back to the mailing-list!

Extending Scapy

If you need to add some new protocols, new functions, anything, you can write it directly into Scapy source file. But this is not very conveniant. Even if those modifications are to be integrated into Scapy, it can be more conveniant to write them in a separate file.

Once you've done that, you can launch Scapy and import your file, but this is still not very conveniant. Another way to do that is to make your file executable and have it call the Scapy function named `interact()`.

```
#! /usr/bin/env python

# Set log level to benefit from Scapy warnings
import logging
logging.getLogger("scapy").setLevel(1)

from scapy.all import *

class Test(Packet):
    name = "Test packet"
    fields_desc = [ ShortField("test1", 1),
                    ShortField("test2", 2) ]

def make_test(x,y):
    return Ether()/IP()/Test(test1=x,test2=y)

if __name__ == "__main__":
    interact(mydict=globals(), mybanner="Test add-on v3.14")
```

If you put the above listing in the `test_interact.py` file and make it executable, you'll get :

```
# ./test_interact.py
Welcome to Scapy (0.9.17.109beta)
Test add-on v3.14
>>> make_test(42,666)
<Ether type=0x800 |<IP |<Test test1=42 test2=666 |>>>
```

Using Scapy in your tools

You can easily use Scapy in your own tools. Just import what you need and do it.

This first example take an IP or a name as first parameter, send an ICMP echo request packet and display the completely dissected return packet.

```
#! /usr/bin/env python

import sys
from scapy.all import sr1,IP,ICMP
```

```
p=sr1(IP(dst=sys.argv[1])/ICMP())
if p:
    p.show()
```

This is a more complex example which does an ARP ping and reports what it found with LaTeX formating.

```
#! /usr/bin/env python
# arping2tex : arpings a network and outputs a LaTeX table as a result

import sys
if len(sys.argv) != 2:
    print "Usage: arping2tex <net>\n  eg: arping2tex 192.168.1.0/24"
    sys.exit(1)

from scapy.all import srp,Ether,ARP,conf
conf.verb=0
ans,unans=srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=sys.argv[1]),
              timeout=2)

print r"\begin{tabular}{|l|l|}"
print r"\hline"
print r"MAC & IP\\"
print r"\hline"
for snd,rcv in ans:
    print rcv.sprintf(r"%Ether.src% & %ARP.psrc%\\")
print r"\hline"
print r"\end{tabular}"
```

Here is another tool that will constantly monitor all interfaces on a machine and print all ARP request it sees, even on 802.11 frames from a Wi-Fi card in monitor mode. Note the `store=0` parameter to `sniff()` to avoid storing all packets in memory for nothing.

```
#! /usr/bin/env python
from scapy.all import *

def arp_monitor_callback(pkt):
    if ARP in pkt and pkt[ARP].op in (1,2): #who-has or is-at
        return pkt.sprintf("%ARP.hwsrc% %ARP.psrc%")

sniff(prn=arp_monitor_callback, filter="arp", store=0)
```

For a real life example, you can check Wifitap