Computer Networks Advanced Course

Scapy

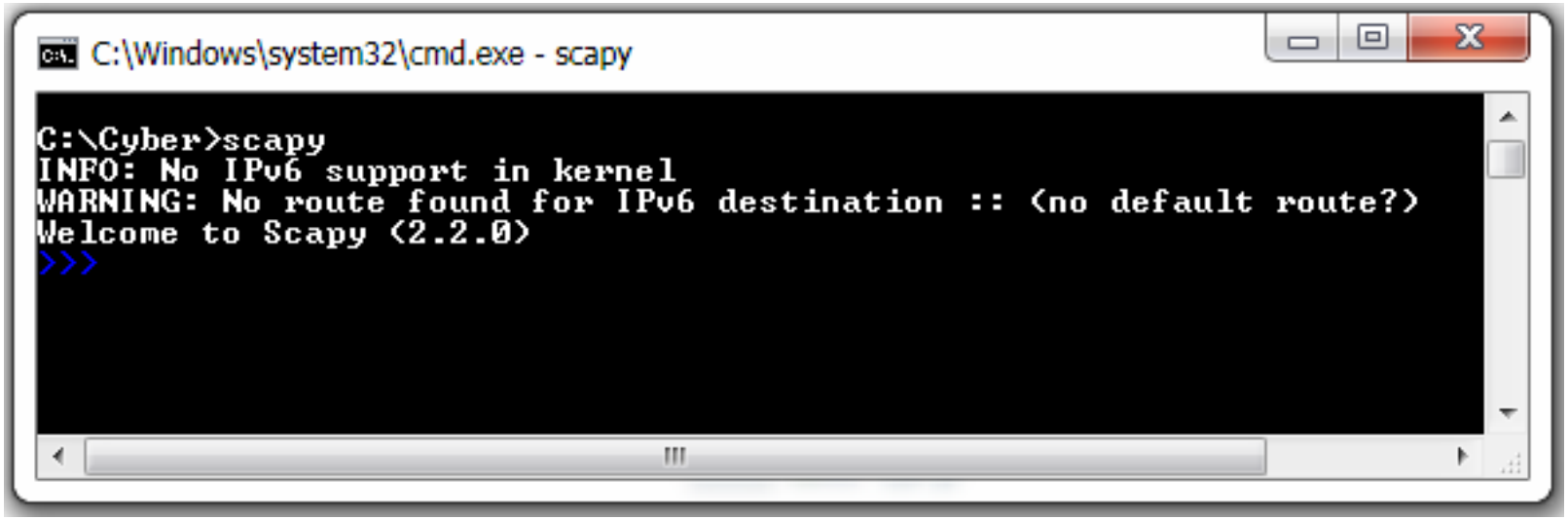
Barak Gonen
Based on Computers Networks
By Rosenboim, Gonen, Hod

Intro

- We learned to program sockets
  - Only application layer
- Scapy:
  - Python import library
  - Sniff packets
  - Craft packets

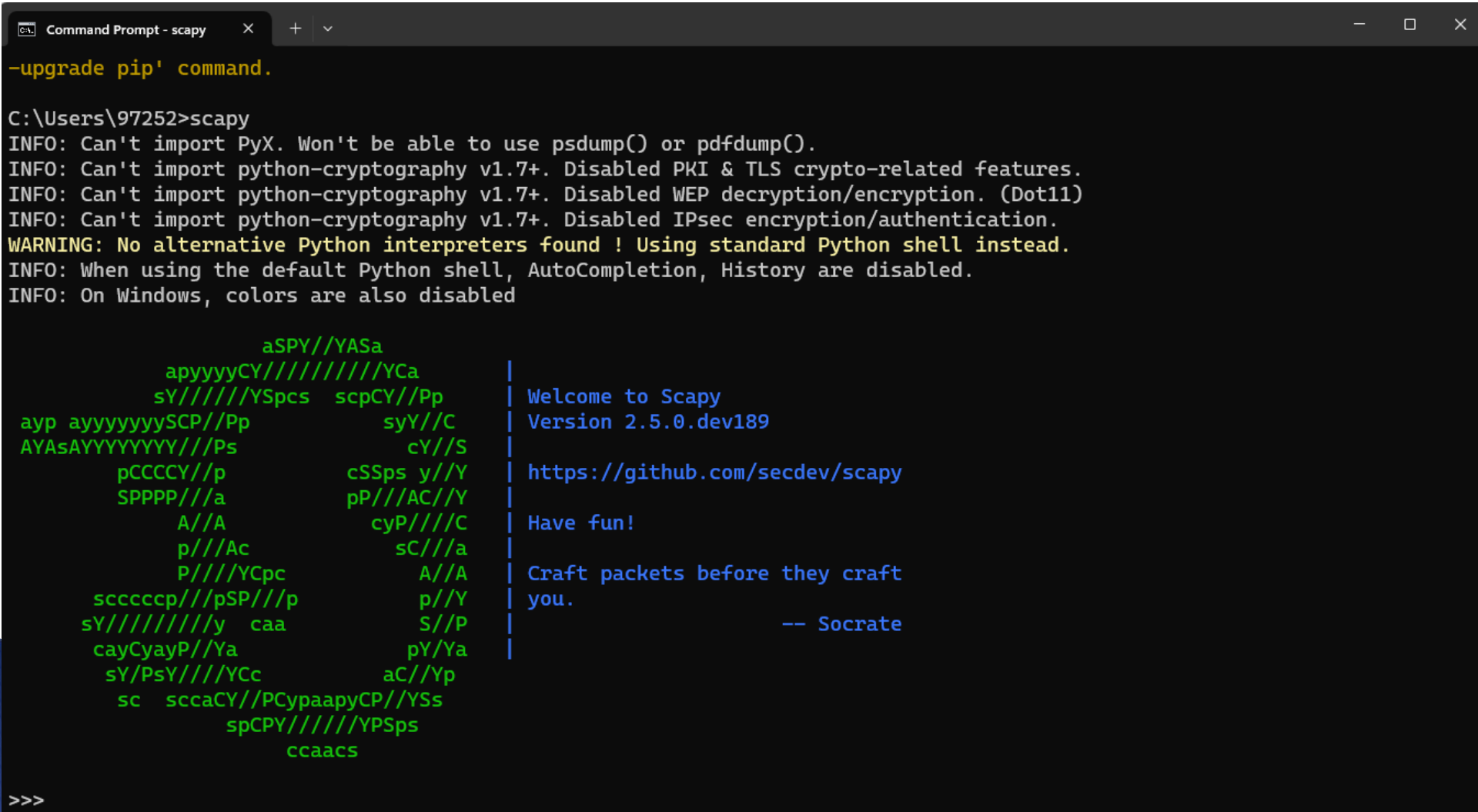
Scapy Fire Up

- CMD -> “scapy”



Scapy Fire Up

- CMD -> “scapy”



Function Sniff

- Let's sniff packets

```
>>> p = sniff(count=2)
>>> p
<Sniffed: TCP:1 UDP:0 ICMP:0 Other:1>
>>>
>>> p.summary()
Ether / ARP who has 192.168.1.175 says 192.168.1.1 / Padding
Ether / IP / TCP 192.168.1.174:61729 > 192.168.1.66:8009 PA / Raw
>>>
```

Function Sniff

- Packets are stored in a list
- Access same as in python

```
>>> p[0]
<Ether  dst=ff:ff:ff:ff:ff:ff src=d4:35:1d:22:47:7e type=ARP |<ARP  hwtype=0x1
 ptype=IPv4 hwlen=6 plen=4 op=who-has hwsrc=d4:35:1d:22:47:7e psrc=192.168.1.1
 hwdst=00:00:00:00:00:00 pdst=192.168.1.175 |<Padding  load='\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' |>>>
>>>
```

Show()

- Nice presentation of packet's fields

```
>>> p[0].show()
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = d4:35:1d:22:47:7e
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = who-has
     hwsrc     = d4:35:1d:22:47:7e
     psrc      = 192.168.1.1
     hwdst     = 00:00:00:00:00:00
     pdst      = 192.168.1.175
###[ Padding ]###
        load      = '\x00\x00\x00\x00\
```

Filter DNS packets (ex 5.1)

- Scapy identifies common protocols
- Custom filtering- use lfilter (small "L")
- Example- DNS filtering

```
>>> def filter_DNS(packet):
...     return DNS in packet
...
>>> p = sniff(count=4, lfilter=filter_DNS)
>>> p.summary()
Ether / IP / UDP / DNS Qry "b'_homekit._tc
Ether / IPv6 / UDP / DNS Qry "b'_homekit._
Ether / IP / UDP / DNS Qry "b'S70PCI-7959a
Ether / IP / UDP / DNS Qry "b'S70PCI-7cad4
>>>
```

Checking Field Values

- The “qr” bit indicates query / response
- There is only DNS Question record (no response), so this packet is a query
- Conclusion: 0 means “Query”

```
>>> p[0][DNS].show()
###[ DNS ]###
  id        = 2
  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.jct.ac.il.lan.'
   |  qtype     = A
   |  qclass    = IN
  an        = None
  ns        = None
  ar        = None

>>>
```

Filter DNS packets

- Suppose we need to filter DNS queries of type A:
  - DNS – learned
  - Query – learned
  - Type A:
    - Do nslookup while scapy-sniffing

```
C:\Users\BARAK>nslookup www.jct.ac.il
```

```
>>> p = sniff(count=4, lfilter=filter_DNS)
>>> p.summary()
Ether / IP / UDP / DNS Qry "b'1.1.168.192.in-addr.arpa.'"
Ether / IP / UDP / DNS Ans "b'OpenWrt.lan.'"
Ether / IP / UDP / DNS Qry "b'www.jct.ac.il.lan.'"
Ether / IP / UDP / DNS Ans
>>>
```

Our query

Filter DNS packets

- Type “A” record

```
>>> p[2][DNSQR].qtype
1
```

- “A” is 1

```
>>> p[2][DNS].show()
###[ DNS ]###
  id        = 2
  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.jct.ac.il.lan.'
   |  qtype     = A
   |  qclass    = IN
  an        = None
  ns        = None
  ar        = None
```

Class Exercise

- Find the value of "CNAME" type DNS packets

```
C:\Users\BARAK>nslookup -type=CNAME www.jct.ac.il
```

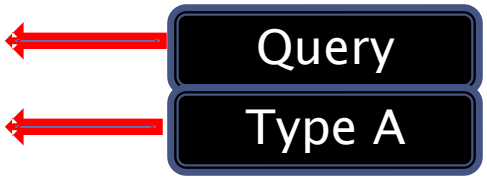
```
>>> p[2][DNSQR].qtype
5
```

```
>>> p[2][DNS].show()
###[ DNS ]###
  id        = 2
  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.jct.ac.il.lan.'
   |  qtype     = CNAME
   |  qclass    = IN
  an        = None
  ns        = None
  ar        = None
```

Filter DNS Packets – cont.

- Refine our filter:

```
>>> def filter_dns(packet):
        if DNS in packet:
            return (packet[DNS].qr==0)
                and (packet[DNSQR].qtype==1)
```



Processing Post Filtering

- Scapy enables processing of filtered packets
- Example- print domain names of DNS queries

```
>>> def print_query_name(dns_packet):
        print(dns_packet[DNSQR].qname)
```

- Use “prn” optional parameter

PRN Parameter

```
>>> p=sniff(count=4, lfilter=filter_DNS_A, prn=print_DNS_qname)
www.jct.ac.il.lan.
www.jct.ac.il.
www.jct.ac.il.lan.
www.jct.ac.il.
>>>
```

- If no results, flush DNS cache
  - ipconfig/flushdns

Interim Summary

- We learned how to use Scapy to sniff packets
- We learned how to process post-filtering
- How about crafting packets?

Nslookup by Scapy

- “Tailor made” packets
- Learning phases:
  - Create packets
  - Add layers
  - Set values
  - Send and receive responses
  - Use in python script

Create “Skeleton” Packet

```
>>> my_packet = IP()
>>> my_packet.show()
```

- Network layer only, IP protocol, default values

```
>>> p = IP()
>>> p.show()
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = ip
  chksum    = None
  src       = 192.168.1.174
  dst       = 127.0.0.1
  \options   \
```

Set Parameters

- Set destination:
  - Modify packet:

```
>>> p[IP].dst = '8.8.8.8'
>>>
```

  - Create new packet:

```
>>> p = IP(dst='8.8.8.8')
>>>
```

  - Note that src automatically set to local IP

```
>>> p.show()
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = ip
  chksum    = None
  src       = 192.168.1.174
  dst       = 8.8.8.8
  \options   \
```

Adding Layers

- Simply write protocol name
- Order of writing- left to right

```
>>> p = IP(dst='8.8.8.8')/UDP()/DNS()
>>>
```

```
>>> p.show()
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = udp
  chksum    = None
  src       = 192.168.1.174
  dst       = 8.8.8.8
  \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   = 0
        ancount   = 0
        nscount   = 0
        arcount   = 0
        qd        = None
        an        = None
        ns        = None
        ar        = None
```

Add Load to Packet

- Let's add Raw over IP packet:

```
>>> p = IP(dst='8.8.8.8')/Raw(load="Bip me!")
>>>
>>> p.show()
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = ip
  chksum    = None
  src       = 192.168.1.174
  dst       = 8.8.8.8
  \options   \
###[ Raw ]###
     load      = 'Bip me!'
```

Command

- Small “hack” to create packets:
- Print(p.command()) will show a scapy command creating identical packet
- Try it on your script which filters DNS requests of type A ☺

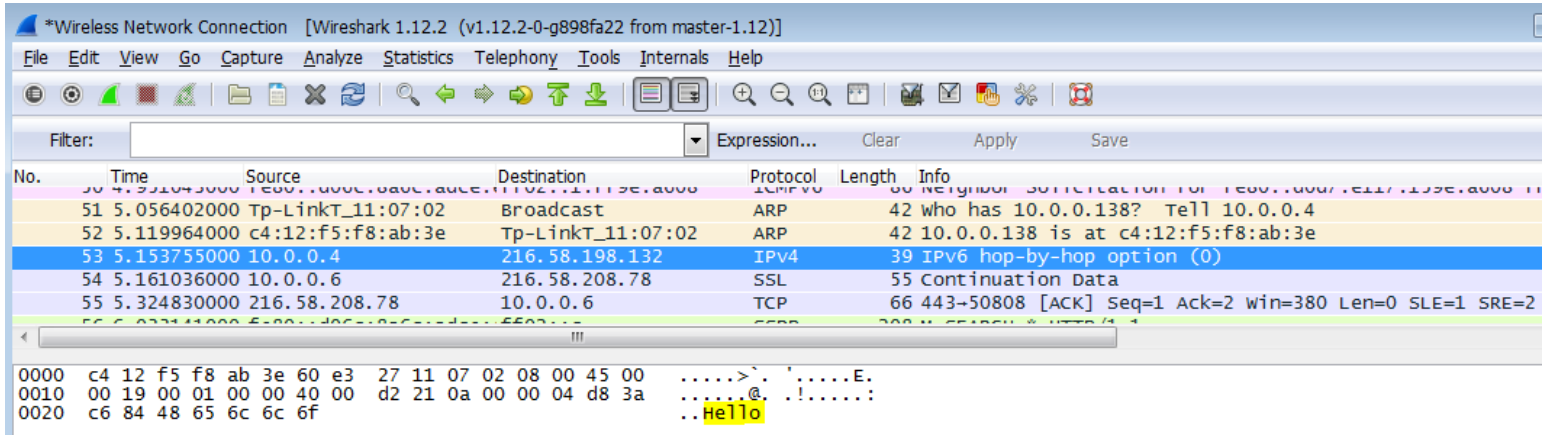
Sending Packets

- Scapy can translate domain name to IP
- Let's load some data and send it to Google:

  >>> my_packet = IP(dst ='www.google.com') / 'Hello'

  >>> send(my_packet)

  - Use Wireshark and find your packet



sr1

- Self study sr1 function
- Textbook ex. 6.10, 6.11
- Craft a DNS packet:
  - DNS server's IP
  - Destination port
  - Flags
  - Requested domain name
  - Query type
- From response packet, extract responses

sr1 – overview

- Function sr1 is like send, but returns the response packet
  - r = sr1(p)
- Example – sending a DNS packet:
  - p = IP(dst="8.8.8.8")/UDP(sport=55555, dport=53) / DNS(rd=1,qdcount=1,qd=DNSQR(qname="www.themarker.com", qtype=1))
  - Or:
  - p = IP(dst="8.8.8.8")/UDP(sport=55555, dport=53) / DNS(rd=1,qdcount=1)/DNSQR(qname="www.themarker.com", qtype=1)
  - r = sr1(p)

Sr1 – cont

- Under the hood –how would scapy filter the response packet from all incoming packets?
- Scapy calculates hash on several fields in request packet, compares them to response packets

```
>>> p.hashret()
b'\xc8\xa0\t\xd5\x11'
>>> r.hashret()
b'\xc8\xa0\t\xd5\x11'
>>>
```

sr1

- Useful parameters:
  - timeout: exit even no packet was received
    - CTRL+C to brute force exit
  - verbose = False : no info printed to screen
- Try it:
  - Create a packet with IP only, dst = “1.2.3.4”
  - Send it using sr1, while making sure there are no prints and the call is not blocking

Nslookup by Scapy

- Extracting numerous DNS responses:
  - Look for ancount value
  - Loop over all indexes in ancount
  - Find if p[DNSRR][i].type has the desired type
- For example:

```
>>> r[DNSRR][1].type
1
```

Sniff not sniffing?

- Either the filter is wrong or the sniffing interface is wrong
  - Disable filter – is anything captured?
- Show_interfaces()
  - Each has index
  - Use ipconfig to check which is active

```
>>> show_interfaces()
INFO: Table cropped to fit the terminal (conf.auto_crop_tables==True)
INDEX  IFACE                                    IP              MAC
-4     [Unknown] Adapter for loopback traff_    127.0.0.1       ff:ff:ff:ff:ff:ff
15     Microsoft Wi-Fi Direct Virtual Adapt_    169.254.13.91   7c:c2:c6:37:75:0c
14     Microsoft Wi-Fi Direct Virtual Adapt_    169.254.16.187  7e:c2:c6:37:75:0c
17     Intel(R) Ethernet Connection (7) I21_    169.254.9.127   2c:f0:5d:79:dc:04
10     TP-Link Wireless USB Adapter             192.168.1.174   7c:c2:c6:37:75:0c
```

Sniff not sniffing?

- Use dev_from_index to configure the interface

```
>>> conf.iface = dev_from_index(17)
```

- Or for only one sniff, use parameter iface:

```
>>> p=sniff(count=1, iface=dev_from_index(11))
```

Fun Exercise

- Make DNS server contact your friend ☺
  - Find a computer on your network
    - Use ipconfig to make sure the default gateway is same
  - Prepare a DNS packet, ip.src is friend's IP
  - Ask your friend to sniff, and sent your packet