TTM4536 - Ethical Hacking - Information Security, Specialization Course

- Plan for 26 Sept 2017

Based on the material of Chapter 4 of the textbook

- The goal is to sniff the traffic with a Python script and the packet manipulation library Scapy

For testing the scripts of Chapter 4 of the textbook make these changes

- For Kali virtual machine change the network adapter to be “Bridged Adapter”
- Change the Promiscuous mode to “Allow All”

What is scapy?

- It's a packet manipulation tool.
- It can forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more.
- Scapy can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery.
- It can replace hping, arpspoof, arp-sk, arping, p0f (and even some parts of Nmap, tcpdump, and tshark).
- Scapy is supported by Unix, Linux, MAC and Windows.

Why scapy?

- Flexible unlike other packet crafting tools with limited functionalities.
- Little knowledge required to build your own tools
- Single Replacement for Multiple tools such as wireshark, nmap, hping etc.
- Build your own tools with Combined Techniques e.g. VLAN hopping + ARP Cache poisoning
- Any field in every TCP/ IP layer can be altered
- Decode packets (Received a TCP Reset on port 80), and not Interprets (Port 80 is Closed)

Basic Commands

- Scapy Start

```
root@bt:~# scapy
INFO: Can't import python gnuplot wrapper . Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
WARNING: No route found for IPv6 destination :: (no default route?)
Welcome to Scapy (2.1.0)
>>>
```

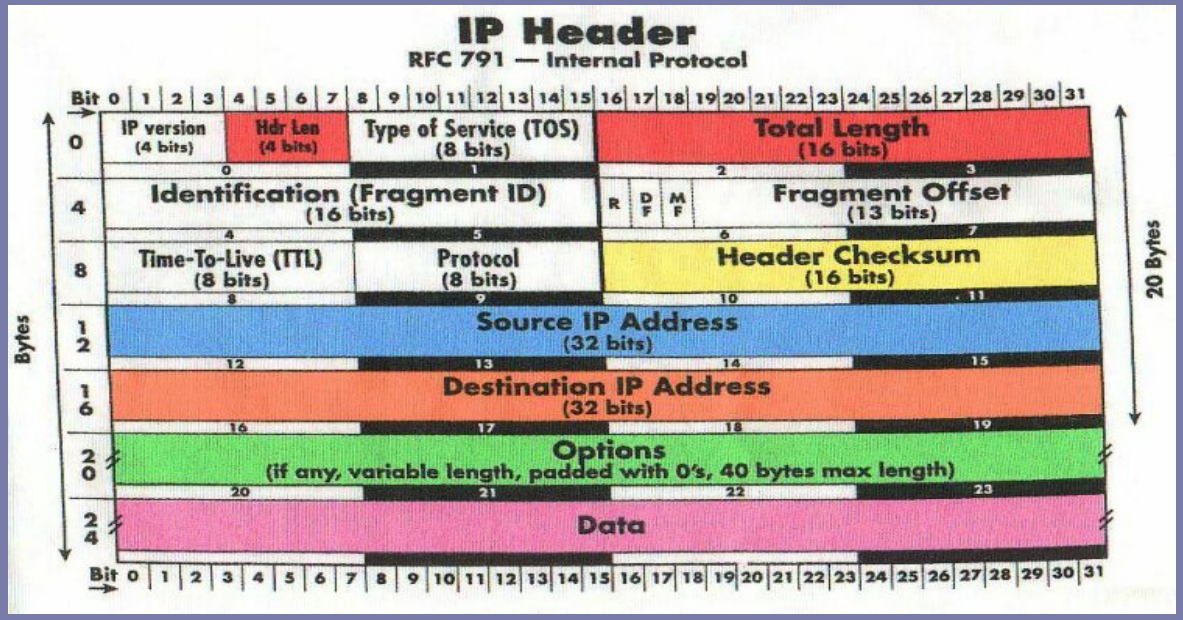
- List of Supported Protocols

```
>>> ls()
ARP         : ARP
ASN1_Packet : None
BOOTP       : BOOTP
CookedLinux : cooked linux
DHCP        : DHCP options
DHCP6       : DHCPv6 Generic Message)
```

- Available Commands in Scapy

```
>>> 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
corrupt_bytes       : Corrupt a given percentage or number of bytes from a strin
g
```

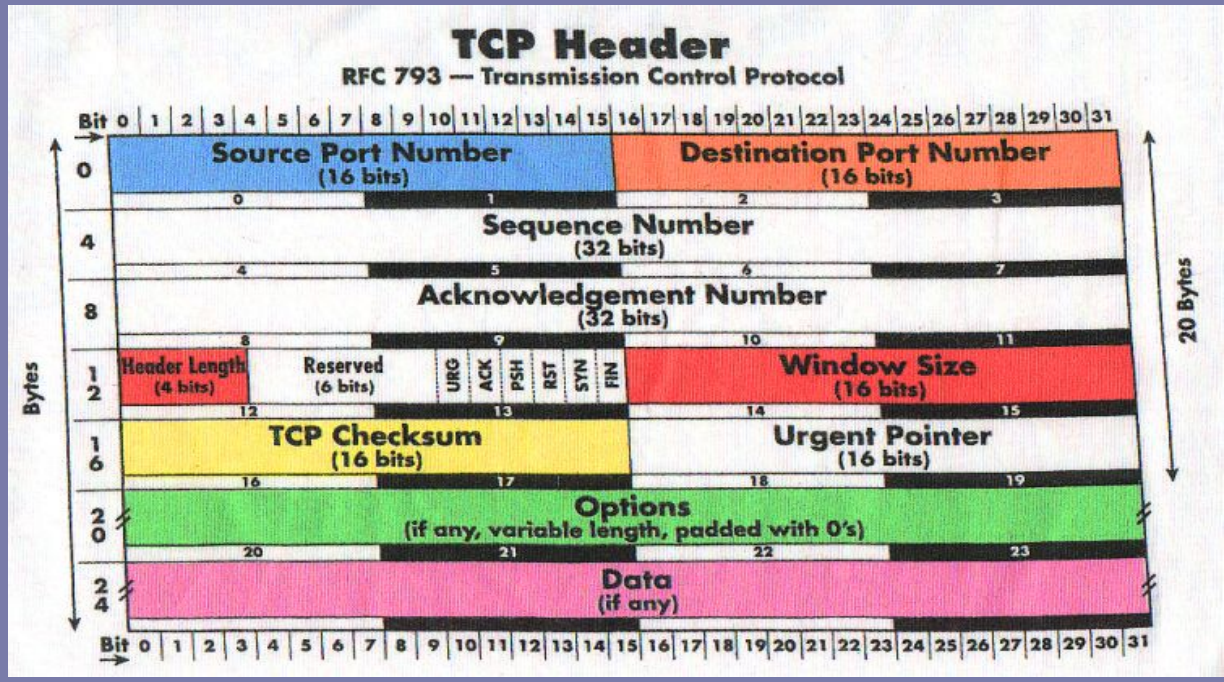
IP Header



IP Fields in Scapy

```
>>> 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      = ([])
```

TCP Header



TCP Fields in Scapy

```
>>> ls(TCP)
sport      : ShortEnumField          = (20)
dport      : ShortEnumField          = (80)
seq        : IntField                = (0)
ack        : IntField                = (0)
dataofs    : BitField                = (None)
reserved   : BitField                = (0)
flags      : FlagsField              = (2)
window     : ShortField              = (8192)
chksum     : XShortField             = (None)
urgptr     : ShortField              = (0)
options    : TCPOptionsField         = ({})
```

Building your first packet

Building packet at IP layer

```
>>> a=IP()
>>> ls(a)
version    : BitField             = 4               (4)
ihl        : BitField             = None            (None)
tos        : XByteField           = 0               (0)
len        : ShortField           = None            (None)
id         : ShortField           = 1               (1)
flags      : FlagsField           = 0               (0)
frag       : BitField             = 0               (0)
ttl        : ByteField            = 64              (64)
proto      : ByteEnumField        = 0               (0)
chksum     : XShortField          = None            (None)
src        : Emph                 = '127.0.0.1'     (None)
dst        : Emph                 = '127.0.0.1'     ('127.0.0.1')
options    : PacketListField      = []              ([])
>>> a.src='10.10.10.1'
>>> a.dst='10.10.10.2'
```

Building packet at TCP layer

```
>>> b=TCP()
>>> ls(b)
sport      : ShortEnumField       = 20              (20)
dport      : ShortEnumField       = 80              (80)
seq        : IntField             = 0               (0)
ack        : IntField             = 0               (0)
dataofs    : BitField             = None            (None)
reserved   : BitField             = 0               (0)
flags      : FlagsField           = 2               (2)
window     : ShortField           = 8192            (8192)
chksum     : XShortField          = None            (None)
urgptr     : ShortField           = 0               (0)
options    : TCPOptionsField      = {}              ({})
>>> b.sport=53
>>> b.dport=[135,139,445,80]
```

Assembling full packet

Assembling full packet at TCP/IP

```
>>> c=a/b
>>> c.show()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= tcp
  chksum= None
  src= 10.10.10.1
  dst= 10.10.10.2
  \options\
###[ TCP ]###
     sport= 1023
     dport= (139, 445)
     seq= 0
     ack= 0
     dataofs= None
     reserved= 0
     flags= S
     window= 8192
     chksum= None
     urgptr= 0
     options= {}
```

Packet ready to send with Calculated values

```
>>> c.show2()
###[ IP ]###
  version= 4L
  ihl= 5L
  tos= 0x0
  len= 40
  id= 1
  flags=
  frag= 0L
  ttl= 64
  proto= tcp
  chksum= 0x52b9
  src= 10.10.10.1
  dst= 10.10.10.2
  \options\
###[ TCP ]###
     sport= 1023
     dport= netbios_ssn
     seq= 0
     ack= 0
     dataofs= 5L
     reserved= 0L
     flags= S
     window= 8192
     chksum= 0x6342
     urgptr= 0
     options= {}
```

Write your own port scanner

Port Scanning :

"An attack that sends client requests to a range of server port addresses on a host, with the goal of finding an active port"

Result Status :

Open : The host sent a reply indicating that a service is listening on the port.

Closed : The host sent a reply indicating that connections will be denied to the port.

Filtered: There was no reply from the host.

Built-in Sniffing Functionality

Sniffing:

"Captures traffic on all or just parts of the network from single machine within the network"

```
root@bt:/# scapy
INFO: Can't import python gnuplot wrapper . Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
WARNING: No route found for IPv6 destination :: (no default route?)
Welcome to Scapy (2.1.0)
>>> sniffed_pkts=sniff(count=10)
```

```
>>> sniffed_pkts.show()
0000 Ether / IP / ICMP 10.10.10.1 > 10.10.10.2 echo-request 0 / Raw
0001 Ether / IP / ICMP 10.10.10.2 > 10.10.10.1 echo-reply 0 / Raw
0002 Ether / IP / ICMP 10.10.10.1 > 10.10.10.2 echo-request 0 / Raw
0003 Ether / IP / ICMP 10.10.10.2 > 10.10.10.1 echo-reply 0 / Raw
0004 Ether / IP / ICMP 10.10.10.1 > 10.10.10.2 echo-request 0 / Raw
0005 Ether / IP / ICMP 10.10.10.2 > 10.10.10.1 echo-reply 0 / Raw
0006 Ether / IP / ICMP 10.10.10.1 > 10.10.10.2 echo-request 0 / Raw
0007 Ether / IP / ICMP 10.10.10.2 > 10.10.10.1 echo-reply 0 / Raw
0008 Ether / IP / ICMP 10.10.10.1 > 10.10.10.2 echo-request 0 / Raw
0009 Ether / IP / ICMP 10.10.10.2 > 10.10.10.1 echo-reply 0 / Raw
```

Installing scapy on Linux

- `sudo apt-get install python-scapy`
- `sudo apt-get install python-gnuplot`
- `sudo apt-get install python-pyx`

Before we play more with scapy let us try several Python scripts

- Open a new project Ch04 in PyCharm
- Before you import and start the three scripts named: mail_sniffer.py, arper.py and pic_carver.py produce a small script mail_sniffer00.py and put the following commands:

mail_sniffer00.py

```
__author__ = 'root'
from scapy.all import *

# our packet callback
def packet_callback(packet):
    print packet.show()
# fire up our sniffer
sniff(prn=packet_callback,count=1)
```

mail_sniffer00.py

Start the scrypt in a terminal with

```
python mail_sniffer00.py
```

mail_sniffer00.py

Start the scrypt in a terminal with

```
python mail_sniffer00.py
```

You should get one sniffed packet. It can be

[ARP] or

[IP] and [UDP] or

[IP] and [TCP] (with a raw packet payload)

mail_sniffer00.py

Start the scrypt in a terminal with

```
python mail_sniffer00.py
```

You should get one sniffed packet. It can be

[ARP] or

[IP] and [UDP] or

[IP] and [TCP] (with a raw packet payload)

Start the script many times to capture all types of packets

mail_sniffer.py

Import the mail_sniffer.py in the project
Start the scrypt in a terminal with

```
python mail_sniffer.py
```

Start an email client with some artificial IP address

The running script mail_sniffer.py should capture the attempts to contact the mail server and will print out the captured IP address, and the packets that have “User” or “Pass” in their payload

mail_sniffer.py

This is just a demo for the sniffing and filtering capability of Scapy

The script will not work as advertised with email clients that establish an encrypted session with TLS

pcap files

- In the field of computer network administration, pcap (packet capture) consists of an application programming interface (API) for capturing network traffic. Unix-like systems implement pcap in the libpcap library; Windows uses a port of libpcap known as WinPcap.
- Monitoring software may use libpcap and/or WinPcap to capture packets travelling over a network and, in newer versions, to transmit packets on a network at the link layer, as well as to get a list of network interfaces for possible use with libpcap or WinPcap.
- The pcap API is written in C, so other languages such as Java, .NET languages, and scripting languages generally use a wrapper; no such wrappers are provided by libpcap or WinPcap itself. C++ programs may link directly to the C API or use an object-oriented wrapper.

arper.py

The goal of this script is to sniff the traffic that is dedicated to a certain IP address with a certain MAC address

Follow the instruction from the textbook

The output is the file arper.pcap

Try to sniff the traffic for a certain machine in the local network (we will try to run also some Windows machine)

The file arper.pcap will be used later for a traffic analysis

arper.py

If you face difficulties to produce a file arper.pcap then try to find some publicly available pcap file on Internet.

pic_carver.py

Rename the file arper.pcap (or the pcap file that you downloaded from Internet) to the file bhp.pcap.

Follow the instructions from the textbook:

- In the current folder of pic_carver.py download the xml script haarcascade_frontalface_alt.xml with the command

  ```
  wget http://eclecti.cc/files/2008/03/haarcascade_frontalface_alt.xml
  ```

- In Kali Terminal run the command

  ```
  apt-get install python-opencv python-numpy python-scipy
  ```

pic_carver.py

Be careful to define proper folders with proper paths in the Python script. For exmple:

```
pictures_directory = "/root/PycharmProjects/Ch04/pic_carver/pictures”
faces_directory    = "/root/PycharmProjects/Ch04/pic_carver/faces"
```

Then with mkdir command in the Terminal create the proper folders

Run the scrypt and see did it find in the file of the captured traffic bhp.pcap some images with human faces

Example of using scapy to solve one Capture The Flag problem

- Hack In The Box 2016 (HITB CTF 2016)
- “Special Delivery” network problem
- You are given a pcap file `'5d176b7cb326f05a1985be5d4d4d9074_special_delivery.pcap'`
- That can be downloaded from the address:

https://github.com/kitctf/writeups/raw/master/hitb2016/special_delivery/5d176b7cb326f05a1985be5d4d4d9074_special_delivery.pcap

- Try to analyze and find the flag.
- The writeup can be found at

https://kitctf.de/writeups/hitbctf/special_delivery

Let's get this solved with scapy. I solved the challenge mostly inside an interactive scapy shell so we'll do the same here:

```
> scapy
>>> pcap = rdpcap('5d176b7cb326f05a1985be5d4d4d9074_special_delivery.pcap')
>>> pcap
<5d176b7cb326f05a1985be5d4d4d9074_special_delivery.pcap: TCP:0 UDP:0 ICMP:267 Other:
41>
```

Alright, let's get rid of the fragments:

```
>>> pcap = defragment(pcap)
>>> pcap
<Defragmented 5d176b7cb326f05a1985be5d4d4d9074_special_delivery.pcap: TCP:0 UDP:0
ICMP:267 Other:0>
```

It seems not every packet has a payload though:

```
>>> pcap[0]
<Ether  dst=00:50:56:e8:3e:63 src=00:0c:29:c2:2c:40 type=0x800 |<IP  version=4L
ihl=5L tos=0x0 len=28 id=22216 flags=DF frag=0L ttl=64 proto=icmp chksum=0x41b0
src=10.13.37.145 dst=192.168.178.34 options=[] |<ICMP  type=echo-request code=0
chksum=0x86e2 id=0x711d seq=0x0 |>>>
>>> pcap[10]
<Ether  dst=00:50:56:e8:3e:63 src=00:0c:29:c2:2c:40 type=0x800 |<IP  version=4L
ihl=5L tos=0x0 len=88 id=22893 flags=DF frag=0L ttl=64 proto=icmp chksum=0x3ecf
src=10.13.37.145 dst=192.168.178.34 options=[] |<ICMP  type=echo-request code=0
chksum=0xa113 id=0x711d seq=0x0 |<Raw
load='E\x00\x00<#\xe1@\x00@\x06\xfc\xd8\n\x00\x03\x02\n\x00\x03\x01\xba\xa1\x00P\xbb
4\xd5\x89\x00\x00\x00\x00\xa0\x02r\x10y\x94\x00\x00\x02\x04\x05\xb4\x04\x02\x08\n\x0
0=\xf6h\x00\x00\x00\x00\x01\x03\x03\n' |>>>>
```

Let's remove the ones that don't have any content:

```
>>> packets = [p for p in pcap if Raw in p]
```

Looking at the ICMP payloads, it looks like there's some binary stuff in every packet… At this point I made a simple guess and assumed that the payload in the ICMP packets were IP packets themselves. We can quickly verify this:

```
>>> ip = IP(packets[0][Raw].load)
>>> ip
<IP  version=4L ihl=5L tos=0x0 len=60 id=9185 flags=DF frag=0L ttl=64 proto=tcp
chksum=0xfcd8 src=10.0.3.2 dst=10.0.3.1 options=[] |<TCP  sport=47777 dport=http
seq=3140801929 ack=0 dataofs=10L reserved=0L flags=S window=29200 chksum=0x7994
urgptr=0 options=[('MSS', 1460), ('SAckOK', ''), ('Timestamp', (4060776, 0)),
('NOP', None), ('WScale', 10)] |>>
```

Ok, this looks like a valid IP packet. Especially the dst and src look good. Let's extract the actual IP packets:

```
>>> packets = PacketList([IP(p[Raw].load) for p in packets])
>>> packets
<PacketList: TCP:175 UDP:0 ICMP:0 Other:0>
```

We could probably do the next steps with Wireshark, but let's write some more python while we're at it.

Quickly scanning the communication (e.g. packets.summary()), it looks like there are only two parties involved. Let's split the packets by source IP (and filter out packets without content again):

```
>>> client_packets = PacketList([p for p in packets if p[IP].src == '10.0.3.2' and
Raw in p])
>>> server_packets = PacketList([p for p in packets if p[IP].src == '10.0.3.1' and
Raw in p])
```

Looking at the first few client packets (e.g. client_packets[:10].hexdump()) shows us that this is an HTTP GET request. Let's dump the response body to a file.

```
>>> response = ''.join(packet[Raw].load for packet in server_packets)
>>> header_length = response.find('\r\n\r\n') + 4
>>> print(response[:header_length])
HTTP/1.1 200 OK
Date: Sun, 21 Feb 2016 16:51:19 GMT
Server: Apache/2.2.22 (Debian)
X-Powered-By: PHP/5.4.4-14+deb7u14
Content-Description: File Transfer
Content-Disposition: attachment; filename="stunnelshell.tgz"
Content-Length: 59913
Expires: 0
Cache-Control: must-revalidate
Pragma: public
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/octet-stream

>>> with open('response', 'w') as f:
...     f.write(response[header_length:])
```

In different shell do the following:

```
> file response
response: gzip compressed data, from Unix, last modified: Sun Feb 21 17:32:52 2016
> tar -xfz response
> ls
response        stunnel         stunnel.pem     stunnel4        stunnelshell.sh
```

Ok, seems like the rest of the communication is encrypted using SSL through stunnel. Luckily we have the certificate (stunnel.pem).. and apparently no forward secrecy was used ;)

We'll now need Wireshark. For that we can either write a new pcap file containing the actual IP traffic:

```
>>> wrpcap('special_delivery.pcap', packets)
```

Or, even simpler, start Wireshark directly from scapy:

```
>>> wireshark(packets)
```

First we'll need to mark the encrypted packets as SSL traffic: Rightclick -> Decode As -> SSL. There's a feature in wireshark for decrypting SSL traffic. It's somewhat hidden though: Preferences -> Protocols -> SSL -> RSA keys list -> Key File. Now all that remains is to rightclick on one of the TLS packets and "Follow [the] SSL Stream":

```
id
uid=0(root) gid=0(root) groups=0(root)
cd /root
cat .flag
HitB{b3a64ecf6978f0593ed20ee15a02ef36}
exit
```

Conclusions

- scapy
  - What is scapy
  - Why it is useful to use scapy
  - How can it be used
- How to sniff packets from python using scapy
- What are pcap files
- How to use scapy to analyze pcap files
- How to solve one CTF problem using scapy