Scapy的基本使用方法

作者：李玉冰

指导老师：杨威

1 概述

> 文档：https://scapy.readthedocs.io/en/latest/introduction.html

`Scapy`是能让用户发送、监听、分析、构造网络数据包的Python程序。`Scapy`是交互式数据包操纵程序，能够轻松处理多种典型任务，如扫描、跟踪路由、探测、单元测试、攻击或网络发现，能够代替`hping`、`arpspoof`、`arpsk`、`arping`、`p0f`以及`Nmap`、`tcpdump`、`tshark`的部分功能。

2 安装

（1）普通安装

```
pip install scapy
```

（2）最新版本

通常使用普通安装的功能已经足够使用，最新版本具体安装方法可参考：https://scapy.readthedocs.io/en/latest/installation.html。

（3）基本使用

Scapy交互式程序就和打开Python命令行导入scapy模块的操作是一样的。

打开交互式Scapy程序，创建一个包`a`，查看字段并发送它的操作如下：

```
$ sudo scapy
Welcome to Scapy (2.2.0-dev)
>>> a = Ether()/IP()/UDP()
>>> a.show()
###[ Ethernet ]###
  dst= ff:ff:ff:ff:ff:ff
  src= 00:00:00:00:00:00
  type= 0x800
###[ IP ]###
     version= 4
     ihl= None
     tos= 0x0
     len= None
     id= 1
     flags=
     frag= 0
     ttl= 64
     proto= udp
     chksum= None
     src= 127.0.0.1
```

```
     dst= 127.0.0.1
     \options\
###[ UDP ]###
        sport= domain
        dport= domain
        len= None
        chksum= None
>>> sendp(a)
.
Sent 1 packets.
```

用Python文件中调用Scapy模块进行同样的操作：

```
# test.py
from scapy.all import *
a = Ether()/IP()/UDP()
sendp(a)
```

输出如下：

```
$ sudo python test.py
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 00:00:00:00:00:00
  type      = 0x800
###[ IP ]###
     version   = 4
     ihl       = None
     tos       = 0x0
     len       = None
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = udp
     chksum    = None
     src       = 127.0.0.1
     dst       = 127.0.0.1
     \options   \
###[ UDP ]###
        sport     = domain
        dport     = domain
        len       = None
        chksum    = None
.
Sent 1 packets.
```

（4）常用命令可以看官方文档`Usage`页面，API查询参考`Scapy API reference`页面。

3 基本使用

3.1 组装一个数据包

组装合法数据包需要按照据包的层级由低层到高层组装，例如UDP数据包，则需要`以太帧-IP头-UDP头-数据`的格式组装，如果只设置一个UDP数据包头部，则会由于没有IP头部导致UDP头部没有校验和。

```
# 不同层用"/"相连接
>>> a = Ether()/IP()/UDP()
```

各层分离地查看数据包a的各个字段，即每个层的头部相互之间没有关联。

```
>>> a.show()
###[ Ethernet ]###
  dst= ff:ff:ff:ff:ff:ff
  src= 00:00:00:00:00:00
  type= 0x800
###[ IP ]###
     version= 4
     ihl= None
     tos= 0x0
     len= None
     id= 1
     flags=
     frag= 0
     ttl= 64
     proto= udp
     chksum= None
     src= 127.0.0.1
     dst= 127.0.0.1
     \options\
###[ UDP ]###
        sport= domain
        dport= domain
        len= None
        chksum= None
```

如果想要修改字段，可以使用`<数据包实例>[数据包头部].<属性名>`来修改各个字段，在各属性不重名时可以省略数据包层级，例

```
>>> a[Ether].dst = "11:22:33:44:55:66"
>>> a[UDP].sport = 1222
>>> a[UDP].dport = 222
```

如果使用`a.dst`，只能修改`Ether()`层，如果修改`a.dst = '192.168.1.8'`，则会将`Ether()`中的`dst`修改，从而导致错误。

观察用`show()`方法得到的结果，可以发现`chksum`是0，这是因为`show()`方法是各层独立地查看字段值。我们可以使用`show2()`方法查看校验和等将各层联系起来的值，它以一个整体的形式看待组装的数据包，所以能够看到计算的校验和，例

```
>>> a.show2()
###[ Ethernet ]###
  dst= 11:22:33:44:55:66
  src= 00:00:00:00:00:00
  type= 0x800
```

```
###[ IP ]###
     version= 4L
     ihl= 5L
     tos= 0x0
     len= 28
     id= 1
     flags=
     frag= 0L
     ttl= 64
     proto= udp
     chksum= 0x7cce
     src= 127.0.0.1
     dst= 127.0.0.1
     \options\
###[ UDP ]###
        sport= 1222
        dport= 222
        len= 8
        chksum= 0xfc37
```

可以看到IP头部的`chksum`和UDP头部的`chksum`都有值。这个UDP数据包没有载荷，我们为它修改源IP和目的IP，再加上数据作为载荷，可以通过输入`a`，来查看数据包`a`的组成：

```
>>> a[Ether].src="11:11:11:00:55:11" # 等同于a.src="11:11:11:00:55:11"
>>> a[IP].dst = "192.168.1.8" # 不能使用a.dst = "..."
>>> a[IP].src = "192.168.1.6"
>>> a = a/"AAAA"
>>> a.show2()
a.show2()
###[ Ethernet ]###
  dst= 11:22:33:44:55:66
  src= 11:11:11:00:55:11
  type= 0x800
###[ IP ]###
     version= 4L
     ihl= 5L
     tos= 0x0
     len= 32
     id= 1
     flags=
     frag= 0L
     ttl= 64
     proto= udp
     chksum= 0xf76d
     src= 192.168.1.6
     dst= 192.168.1.8
     \options\
###[ UDP ]###
        sport= 1222
        dport= 222
        len= 12
        chksum= 0xf450
###[ Raw ]###
           load= 'AAAA'
>>> a
```

```
<Ether  dst=11:22:33:44:55:66 src=11:11:11:00:55:11 type=0x800 |<IP  frag=0
proto=udp src=192.168.1.6 dst=192.168.1.8 |<UDP  sport=1222 dport=222 |<Raw
 load='AAAA' |>>>>
```

可以在组装数据包的时候直接指定各字段的值，在下面的例子中`b`是和`a`完全相同的数据包：

```
b =
Ether(dst="11:22:33:44:55:66",src="11:11:11:00:55:11")/IP(src="192.168.1.6",dst=
"192.168.1.8")/UDP(sport=1222,dport=222)/"AAAA"
>>> b
<Ether  dst=11:22:33:44:55:66 src=11:11:11:00:55:11 type=0x800 |<IP  frag=0
proto=udp src=192.168.1.6 dst=192.168.1.8 |<UDP  sport=1222 dport=222 |<Raw
 load='AAAA' |>>>>
```

注：Scapy支持多种协议，具体支持的内容（每种协议/报文的类名称）可以将协议/协议字段作为关键词，在官方文档的API（`Scapy API Reference`）中进行搜索，例：组装一个TCP数据包：`a=Ether()/IP()/TCP()`。（NS请求报文：`ICMPv6ND_NS`、NA应答报文：`ICMPv6ND_NA`）。

`ls(<数据包实例>)`可以查看该数据包对应字段的类型、当前值、默认值：

```
>>> ls(a)
dst        : DestMACField         = '11:22:33:44:55:66' (None)
src        : SourceMACField       = '11:11:11:00:55:11' (None)
type       : XShortEnumField      = 2048            (0)
--
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        = 17              (0)
chksum     : XShortField          = None            (None)
src        : Emph                 = '192.168.1.6'   (None)
dst        : Emph                 = '192.168.1.8'   ('127.0.0.1')
options    : PacketListField      = []              ([])
--
sport      : ShortEnumField       = 1222            (53)
dport      : ShortEnumField       = 222             (53)
len        : ShortField           = None            (None)
chksum     : XShortField          = None            (None)
--
load       : StrField             = 'AAAA'          ('')
```

3.2 发送数据包

发送数据包有两种基本方法：`send()`和`sendp()`，使用`send()`时从IP层开始构造，使用`sendp()`时从Ether层开始构造。

- `send(x, iface=None, **kargs)`：`x`：packet(s)，`iface`：发送数据包的网络接口（网卡名），在3层发包，程序自动处理链路层上面的 mac 地址，iface表示能够连接到外部网络设备的三层网络接口，如`ens33`。

`192.168.100.5`向`192.168.100.7`发送下面的IP包，可以在`192.168.100.5`的`ens33`接口监听到这个数据包，也可以在`192.168.100.7`的`ens160`接口上监听到这个数据包。

```
a = IP(dst='192.168.100.5')
send(a, iface="ens33")
```

注：网络接口名称可以通过`ifconfig`命令在Linux系统上查看。

构造一个UDP数据包，比较加上以太头和不加以太头结果的区别。

```
a = IP(dst='192.168.100.5')/UDP(sport=666,dport=222)/"test"
send(a, iface="ens33")

b = Ether()/IP(dst='192.168.100.5')/UDP(sport=666,dport=222)/"test"
send(b, iface="ens33")
```

结果是数据包a可以成功在`192.168.100.5`接收，数据包b不可以。

数据包b可能由于错误的MAC地址无法成功发送，根据`ifconfig`命令修改数据包b中网卡接口的MAC地址，再进行发送，也无法送达。

所以应注意使用`send()`时，不要加`Ether()头`。

- `sendp(x, iface=None, iface_hint=None, socket=None, **kargs)`：`x`：packet(s)，`iface`：发送数据包的接口，为2层网络接口，如交换机虚拟出来的2层接口`veth2`，在2层发包。

  构造一个UDP数据包，和`send()`测试结果相反地，用`sendp()`发送数据包时必须加上`Ether()`，否则数据包无法发送到目的地。`192.168.100.5`能够收到数据包b，而无法收到数据包a。

3.3 从文件读包并发送

从pcap文件中读取数据包到变量中，依次发送。

```
a = rdpcap("icmpv6_small.pcap")
iface = "veth0"
sendp(a, iface)
```

3.4 自定义协议的数据包

创建`Packet`的子类，然后设置它的层关系，例如下面定义了一个计算协议，当以太类型为`0x1234`时，上层协议是`p4calc`。

```
from scapy.all import *

class P4calc(Packet):
    name="P4 Calculator"
    fields_desc=[
        StrFixedLenField('P','P',1),
        StrFixedLenField('Four', '4', 1),
        ByteField('Version', 1),
        StrFixedLenField('Op', '\x00', 1),
        ByteField('Opcode', '\x00', 2),
```

```
        IntField('OperA', 0),
        IntField('OperB', 0),
        IntField('Result', 0)]

bind_layers(Ether, p4calc, type=0x1234)
p = (Ether(dst="00:11:22:33:44:55", src="00:aa:bb:cc:dd:ee")/
     P4calc(Opcode="\x01", Op="+", OperA=10, OperB=20)
```

3.5 以指定速度发包

> 文档：https://scapy.readthedocs.io/en/latest/api/scapy.sendrecv.html

（1）低速发包

在3.2中我们仅设置了`send`和`sendp`方法的两个参数，即发送的数据包列表（list）`x`和发送的网络接口`iface`，但实际上`send`和`sendp`方法还有下面几个常用参数：

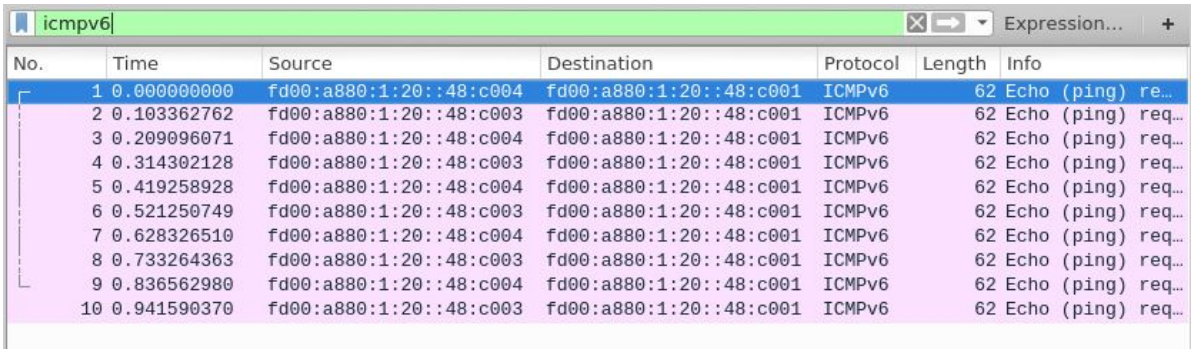
- **x** – 待发送的数据包/数据包列表。
- **inter** – 发送两个数据包的间隔，单位为秒，默认值为0。
- **loop** – 默认值为0，不为0则循环发送`x`，按`Ctrl+c`停止。
- **count** – 计划发送`x`的次数，默认为发送1次。
- **iface** – 发送数据包的网络接口。

在`预期数据包速度较低`时，可以使用`inter`参数，设置两个两个数据包的发送间隔。

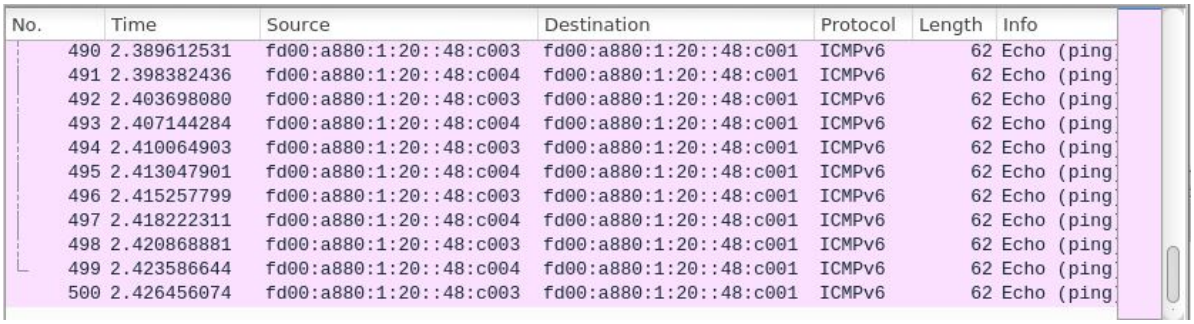
下面的程序构造了一个包含2个ICMPv6 Echo Request数据包的数据包列表，从`ens33`网口循环发送这个数据包列表，总共发送5次，发送间隔为0.1s。

```
from scapy.all import *

print("Sending ICMPv6 Echo Request packets...")

p1 = (Ether(dst="00:11:22:33:44:55", src="00:aa:bb:cc:dd:e1")/

IPv6(nh=58,src="FD00:A880:0001:0020:0000:0000:0048:C004",dst="FD00:A880:0001:002
0:0000:0000:0048:C001")/
     ICMPv6EchoRequest())

p2 = (Ether(dst="00:11:22:33:44:55", src="00:aa:bb:cc:dd:f1")/

IPv6(nh=58,src="FD00:A880:0001:0020:0000:0000:0048:C003",dst="FD00:A880:0001:002
0:0000:0000:0048:C001")/
     ICMPv6EchoRequest())

p = [p1,p2]

sendp(p, iface="ens33", loop=1, inter=0.001, count=5)
```

我们运行这个程序，并开启wireshark监听`ens33`收到的`icmpv6`数据包，得到的结果如下图：



可以看到我们收到了10个数据包，其数据包的发送间隔约为0.1s。我们将`inter`参数修改为`0.001`，`count`参数修改为250，再用wireshark监听`ens33`，得到结果如下图：



理论上应该用0.5s发完，但实际上我们用了2.42s，用`sendp`中的`inter`控制发包速度和我们的预期并不相符。如果直接构造一个包含1000个数据包的列表用`sendp`发送，也不能用0.5s发完。

(2) 高速发包

高速发包用`sendpfast`，需要安装`tcpreplay`，安装方法如下。

ubuntu安装`tcpreplay`：

```
apt-get install tcpreplay
```

`sendpfast`的常用参数如下：

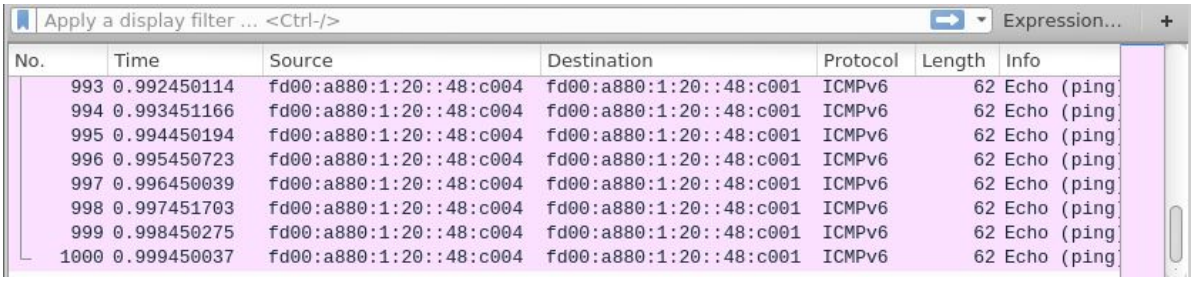
- **x** – 待发送的数据包/数据包列表。
- **pps** – 每秒发送的数据包数。
- **mpbs** – 每秒发送的MBits。
- **loop** – 发送数据包列表的进程数，它的值相当于`send`中的`count`，不同的地方在于它发送一次会重新创建一个发包进程，进程的启动速度很慢，高速发包时应将loop设置为1。另外它的默认值为0，如果不设置此参数会发送0次`x`。
- **iface** – 发送数据包的网络接口。

下面的程序先构造了一个包含1000个数据包的列表，然后将每秒发送的数据包设置为1000。

```
from scapy.all import *
print("Sending ICMPv6 Echo Request packets...")

p1 = (Ether(dst="00:11:22:33:44:55", src="00:aa:bb:cc:dd:e1")/

IPv6(nh=58,src="FD00:A880:0001:0020:0000:0000:0048:C004",dst="FD00:A880:0001:002
0:0000:0000:0048:C001")/
     ICMPv6EchoRequest())

p2 = (Ether(dst="00:11:22:33:44:55", src="00:aa:bb:cc:dd:f1")/
```

```
IPv6(nh=58,src="FD00:A880:0001:0020:0000:0000:0048:C003",dst="FD00:A880:0001:002
0:0000:0000:0048:C001")/
     ICMPv6EchoRequest())

p = []
for i in range(1000):
    p.append(p1)
sendpfast(p, pps=1000, iface='ens33', loop = 1)
```

我们运行这个程序并用wireshark监听`ens33`，得到结果如下图。



从图中我们可以看到发送1000个数据包大约用了1s，和我们的预期相符。

3.6 监听数据包

> 文档：https://scapy.readthedocs.io/en/latest/api/scapy.sendrecv.html

在Scapy中，可以使用`sniff`函数调用抓包分析，并对抓到的包进行回调操作。`sniff`的详细参数可以查询文档，这里仅介绍几个基本参数：

- **count** – 捕获的数据包数量，0表示一直捕获。
- **prn** – 对每个捕获的数据包进行的操作函数，函数的输入是捕获的数据包。
- **filter** – 过滤规则，语法同wireshark。
- **iface** – 指定抓包的网卡/网卡列表，不指定则监听所有网卡。

下面的程序监听8个收到的ipv6数据包，并输出它们的目的IPv6地址

```
# sniff.py
from scapy.all import *

def pack_callback(packet):
    ipv6_addr = packet['IPv6'].dst
    print(ipv6_addr)

def start_sniff():
    print("Start Listen ...")
    filterstr="ip6"
    sniff(filter=filterstr,prn=pack_callback, iface='ens33', count=8)

if __name__ == '__main__':
    start_sniff()
```

我们先运行`sniff.py`监听网卡，然后用下面的程序发送10个IPv6数据包，查看运行结果。

```
# sniff_test.py
from scapy.all import *
print("Sending ICMPv6 Echo Request packets...")

p = []
for i in range(10):
    pkt = (Ether(dst="00:11:22:33:44:55", src="00:aa:bb:cc:dd:e1")/

IPv6(nh=58,src="FD00:A880:0001:0020:0000:0000:0048:C004",dst="FD00:A880:0001:002
0:0000:0000:0048:C00"+str(i))/
          ICMPv6EchoRequest())
    p.append(pkt)

sendpfast(p, pps=1000, iface='ens33', loop = 1)
```

运行结果如下，可以看到`sniff.py`监听了8个数据包。

```
Start Listen ...
fd00:a880:1:20::48:c000
fd00:a880:1:20::48:c001
fd00:a880:1:20::48:c002
fd00:a880:1:20::48:c003
fd00:a880:1:20::48:c004
fd00:a880:1:20::48:c005
fd00:a880:1:20::48:c006
fd00:a880:1:20::48:c007
```

4 总结

以上就是我在P4-NSAF小论文中涉及到的Scapy基本操作，希望能够对大家有帮助。

参考资料

1.Scapy文档：https://scapy.readthedocs.io/en/latest/introduction.html

2.Scapy Git：https://github.com/secdev/scapy

3.协议头scapy.layers.inet6：https://scapy.readthedocs.io/en/latest/api/scapy.layers.inet6.html?highlight=ICMPv6EchoRequest

4.Python3下基于Scapy库完成网卡抓包解析：https://cloud.tencent.com/developer/article/1694737