网络测量第二次实验报告

实验背景

一、网络测量的分类
根据测量方式不同，网络测量可分为主动测量和被动测量。
①主动测量
为了测量网络而主动产生新的用于测量的网络流量而注入网络，并根据测量数据流的传送情况来分析网络性能、状态等参数的一类测量方法。
②被动测量
不向网络发送数据包，而是监听网络中的流量来进行网络的测量，被动测量不会增加网络的负载，但它部署起来却相对困难，一般需要配有专用设备或拥有特殊权限，这对于普通用户来说不太容易实现。

二、Scapy
Scapy 是一个 Python 第三方库，用户能够基于 Scapy 发送，嗅探和剖析并伪造网络数据包。此功能使用户能够构建探测，扫描或攻击网络的工具。
Scapy 是一个功能强大的交互式数据包操作程序。它能够伪造或解码大量协议的数据包，通过线路发送，捕获，匹配请求和回复等等。Scapy 可以轻松处理大多数经典任务，如扫描，跟踪路由，探测，单元测试，攻击或网络发现。它可以取代 hping，arpspoof，arp-sk，arping，p0f 甚至是 Nmap，tcpdump 和 tshark 的某些部分。

实验内容

1.在 scapy 的四种经典应用中（端口扫描、主机扫描、路由跟踪、模拟攻击）任选三种场景进行尝试，如 TCP ACK 扫描、ICMP 主机发现、DNS 放大攻击三种。
2.程序满足一定的性能需求，最好使用多线程进行，给出具体的程序运行消耗的时间。
3.尽量考虑多种输入，有一定的输入鲁棒性。如输入的是单个 ip 还是网段，输入的是包含特定端口的列表还是一个端口范围，能够做成参数列表形式最好。

项目结构

本次实验选择端口扫描、主机扫描、路由跟踪三种场景。
程序主要的函数结构如下：

```
lab2.py
    SYN_scan(ip, port)//TCP SYN 扫描
    ACK_scan(ip, port)//TCP ACK 扫描
    ARP_host(ip, port)//ARP 主机发现
    ICMP_host(ip, port)//ICMP 主机发现
    trace(ip, port)//路由跟踪

    main()//主函数
```

使用说明

本项目实验环境为 Win10，python 版本为 3.7.0，安装 scapy 库。

```
pip install scapy
```

命令行参数如下：

```
python lab2.py -m <mode> -i <target host> -n <website> -p <target port>
mode:
1 代表选择 TCP SYN 扫描
2 代表选择 TCP ACK 扫描
3 代表选择 ARP 扫描
4 代表选择 ICMP 扫描
5 代表选择路由跟踪
```

实验代码说明

1、ACK_scan(ip, port)

根据参数中的 ip 地址和端口号构建 ACK 数据包并发送，收取到返回值时判断结果。

2、SYN_scan(ip, port)

根据参数中的 ip 地址和端口号构建 SYN 数据包并发送，收取到返回值时判断结果。

3、ARP_host(ip, port)

根据参数中的 ip 地址构建 ARP 数据包并发送，收取到返回值时判断结果。

4、ICMP_host(ip, port)

根据参数中的 ip 地址构建 ICMP 数据包并发送，收取到返回值时判断结果。

5、trace(ip, port)

直接调用 scapy 库中的 traceroute 函数进行路由追踪

6、main()

根据命令行输入的命令获取 ip 地址、端口号或网段等信息，并进行 mode 的选择。对命令进行划分完成后，调用相应的函数进行主动测量。同时采用多线程进行操作。

```
    for p in tgtPorts:
        port = int(p)
            if Mode == 1:
                t = Thread(target=SYN_scan, args=(tgtIP, port))
                t.start()
            if Mode == 2:
                t = Thread(target=ACK_scan, args=(tgtIP, port))
                t.start()
            if Mode == 3:
                t = Thread(target=ARP_host, args=(tgtIP, port))
                t.start()
            if Mode == 4:
                t = Thread(target=ICMP_host, args=(tgtIP, port))
```

```
            t.start()
        if Mode == 5:
            t = Thread(target=trace, args=(tgtIP, port))
            t.start()
```

程序运行结果

1、进行 TCP SYN 扫描

```
F:\experiment2>python lab2.py -m 1 -i www.baidu.com -p 80
WARNING: Mac address to reach destination not found. Using broadcast.
www.baidu.com TCP 80 open
```

31	13.611554	10.208.69.36	112.80.248.76	TCP	54	20 → 80 [SYN] Seq=0 Win=8192 Len=0
32	13.618467	112.80.248.76	10.208.69.36	TCP	60	80 → 20 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0

2、进行 TCP ACK 扫描

```
F:\experiment2>python lab2.py -m 2 -i www.baidu.com -p 80
WARNING: Mac address to reach destination not found. Using broadcast.
www.baidu.com port 80 is unfiltered.
```

603	160.645645	10.208.69.36	36.152.44.96	TCP	54	20 → 80 [FIN] Seq=1 Win=8192 Len=0

3、进行 ARP 扫描

```
F:\experiment2>python lab2.py -m 3 -i www.baidu.com -p 80
[+] Scaned 1 host
Host        MAC
112.80.248.76       00:00:5e:00:01:01
```

1018	291.953497	LiteonTe_ef:c5:f9	Broadcast	ARP	42	Who has 180.101.49.12? Tell 10.208.69.36
1019	291.955489	IETF-VRRP-VRID_01	LiteonTe_ef:c5:f9	ARP	56	180.101.49.12 is at 00:00:5e:00:01:01

4、进行 ICMP 扫描

```
F:\experiment2>python lab2.py -m 4 -i www.baidu.com -p 80
Begin emission:
.............*.....................................*......*..........*..........*..
.....................*..*.....*.............*.........*......*.............*......
.....................*..*.Finished sending 256 packets.
.........**...**********.
Received 296 packets, got 35 answers, remaining 221 packets
```

3368	278.328208	10.208.69.36	10.208.69.1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
3387	280.329002	10.208.69.36	10.208.69.2	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
3410	282.336984	10.208.69.36	10.208.69.3	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
3427	284.338923	10.208.69.36	10.208.69.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
3448	286.353818	10.208.69.36	10.208.69.5	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
3474	288.358171	10.208.69.36	10.208.69.6	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
3477	288.371035	10.208.69.36	10.208.69.7	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
3480	288.383780	10.208.69.36	10.208.69.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
3498	290.391737	10.208.69.36	10.208.69.9	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
3516	292.392834	10.208.69.36	10.208.69.10	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
3530	294.407951	10.208.69.36	10.208.69.11	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
3548	296.410328	10.208.69.36	10.208.69.12	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
3560	298.425099	10.208.69.36	10.208.69.13	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 3562)

5、进行路由跟踪

```
F:\experiment2>python lab2.py -m 5 -i www.baidu.com -p 80
Begin emission:
Finished sending 30 packets.
*.*.*.*.**.**********
Received 31 packets, got 26 answers, remaining 4 packets
   180.101.49.12:tcp80
2  10.80.128.141    11
3  10.80.128.149    11
4  10.80.3.10       11
5  180.101.230.145  11
6  221.231.175.33   11
8  58.213.94.54     11
10 58.213.96.110    11
11 10.166.50.6      11
12 10.166.50.8      11
14 180.101.49.12    SA
15 180.101.49.12    SA
16 180.101.49.12    SA
17 180.101.49.12    SA
18 180.101.49.12    SA
19 180.101.49.12    SA
20 180.101.49.12    SA
21 180.101.49.12    SA
22 180.101.49.12    SA
23 180.101.49.12    SA
24 180.101.49.12    SA
25 180.101.49.12    SA
26 180.101.49.12    SA
27 180.101.49.12    SA
28 180.101.49.12    SA
29 180.101.49.12    SA
30 180.101.49.12    SA
dict_keys(['180.101.49.12'])
```

10113	935.651394	10.208.69.36	180.101.49.12	TCP	54 30411 → 80 [SYN] Seq=0 Win=8192 Len=0
10114	935.657398	10.208.69.36	180.101.49.12	TCP	54 193 → 80 [SYN] Seq=0 Win=8192 Len=0
10115	935.660453	180.101.49.12	10.208.69.36	TCP	60 80 → 30411 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
10116	935.662897	10.208.69.36	180.101.49.12	TCP	54 3905 → 80 [SYN] Seq=0 Win=8192 Len=0
10117	935.666619	180.101.49.12	10.208.69.36	TCP	60 80 → 193 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
10118	935.668359	10.208.69.36	180.101.49.12	TCP	54 65440 → 80 [SYN] Seq=0 Win=8192 Len=0
10119	935.671405	180.101.49.12	10.208.69.36	TCP	60 80 → 3905 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
10120	935.673978	10.208.69.36	180.101.49.12	TCP	54 40190 → 80 [SYN] Seq=0 Win=8192 Len=0
10121	935.677097	180.101.49.12	10.208.69.36	TCP	60 80 → 65440 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
10122	935.679734	10.208.69.36	180.101.49.12	TCP	54 33081 → 80 [SYN] Seq=0 Win=8192 Len=0
10123	935.679939	180.101.49.12	10.208.69.36	TCP	60 80 → 40190 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
10124	935.684912	180.101.49.12	10.208.69.36	TCP	60 80 → 33081 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
10125	935.685863	10.208.69.36	180.101.49.12	TCP	54 58246 → 80 [SYN] Seq=0 Win=8192 Len=0
10126	935.692127	10.208.69.36	180.101.49.12	TCP	54 7349 → 80 [SYN] Seq=0 Win=8192 Len=0
10127	935.694803	180.101.49.12	10.208.69.36	TCP	60 80 → 58246 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
10128	935.697575	10.208.69.36	180.101.49.12	TCP	54 15529 → 80 [SYN] Seq=0 Win=8192 Len=0
10129	935.701617	180.101.49.12	10.208.69.36	TCP	60 80 → 7349 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
10130	935.702606	10.208.69.36	180.101.49.12	TCP	54 27999 → 80 [SYN] Seq=0 Win=8192 Len=0
10131	935.706572	180.101.49.12	10.208.69.36	TCP	60 80 → 15529 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
10132	935.707992	10.208.69.36	180.101.49.12	TCP	54 60480 → 80 [SYN] Seq=0 Win=8192 Len=0
10133	935.710953	180.101.49.12	10.208.69.36	TCP	60 80 → 27999 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
10134	935.713645	10.208.69.36	180.101.49.12	TCP	54 41744 → 80 [SYN] Seq=0 Win=8192 Len=0

总结

这次实验我们学会了运用 python 的 scapy 库以及发包操作、分析数据包等内容，熟悉了相关的操作，基本满足了实验要求。但由于网络环境问题和对 python 多线程的不熟悉，没能实现计算程序运行消耗的时间。