《扫描探测工具识别》

第7组汇报

汇报人：赵奕翔

小组成员：畅晨铭

田　婕

导师：周　舟

时间：2021年8月25日

目录

contents

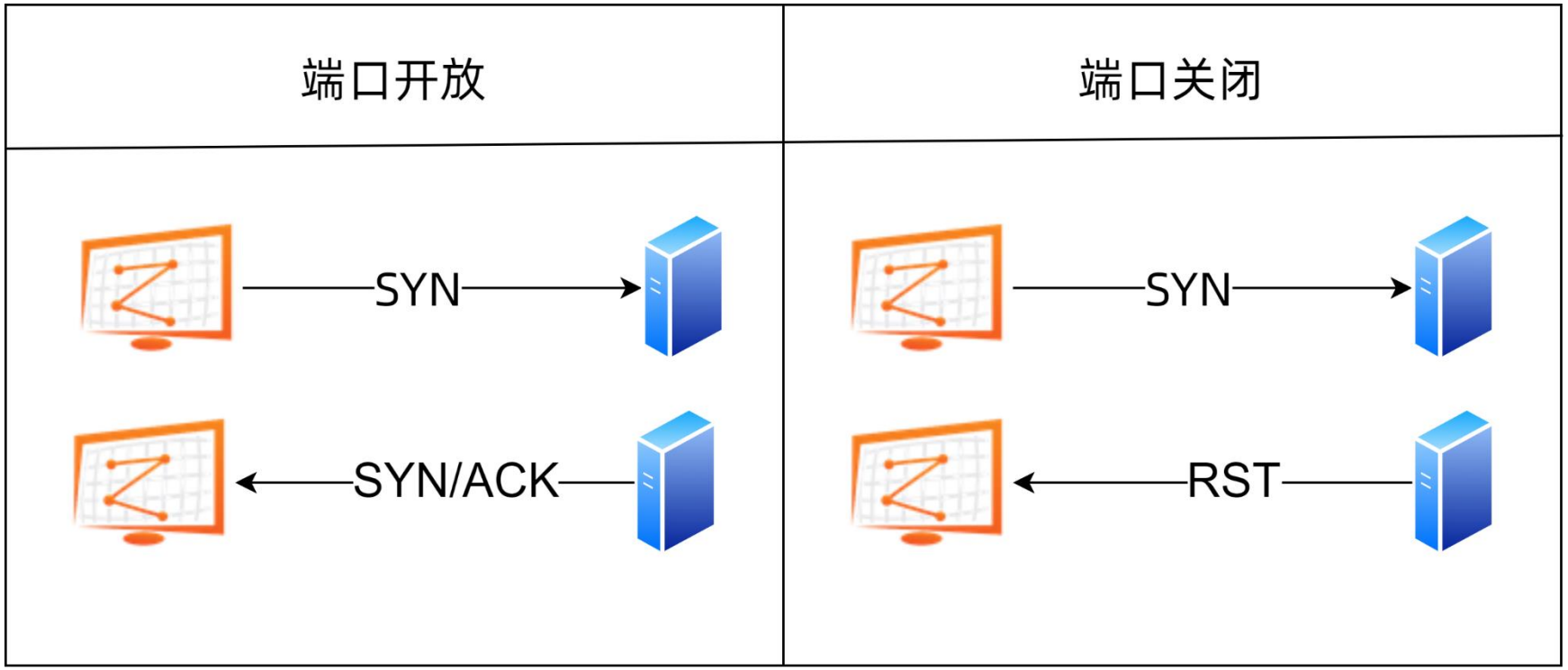
抓包分析

抓包分析

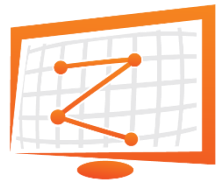
扫描探测工具	类别	实验环境	抓包工具
Zmap	主流扫描工具	Ubuntu18.04 LTS	Wireshark
Angry IP Scanner	端口扫描工具	Windows10	
X-Scan -> Masscan	主流扫描工具	Ubuntu18.04 LTS	

抓包分析——Zmap

ZMap 被设计用来针对整个 IPv4 地址空间或其中的大部分实施综合扫描的工具。
在一台拥有千兆连接的计算机上，ZMap可以在45分钟内扫描整个公共IPv4地址空间。
使用万兆连接和PF_RING，ZMap可以在5分钟内扫描整个IPv4地址空间。



抓包分析——Zmap



较为保守的情况下，对 10,000 个随机的地址的 80 端口以 10Mbps 的速度扫描。

--bandwidth=10M　　--target-port=80　　--max-targets=10000

```
root@ubuntu:/data/zmaplab# zmap --bandwidth=10M --target-port=80 --max-targets=10000 --output-file=results.csv
Aug 19 22:47:42.760 [INFO] zmap: output module: csv
 0:00 0%; send: 0 0 p/s (0 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
 0:00 0%; send: 2 5.51 Kp/s (61 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
 0:01 12%; send: 10000 done (14.2 Kp/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
 0:02 23%; send: 10000 done (14.2 Kp/s avg); recv: 14 13 p/s (6 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.14%
 0:03 35%; send: 10000 done (14.2 Kp/s avg); recv: 14 0 p/s (4 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.14%
 0:04 46%; send: 10000 done (14.2 Kp/s avg); recv: 14 0 p/s (3 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.14%
 0:05 58% (4s left); send: 10000 done (14.2 Kp/s avg); recv: 14 0 p/s (2 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.14
%
 0:06 69% (3s left); send: 10000 done (14.2 Kp/s avg); recv: 14 0 p/s (2 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.14
%
 0:07 81% (2s left); send: 10000 done (14.2 Kp/s avg); recv: 14 0 p/s (1 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.14
%
 0:08 92% (1s left); send: 10000 done (14.2 Kp/s avg); recv: 14 0 p/s (1 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.14
%
Aug 19 22:47:51.799 [INFO] zmap: completed
root@ubuntu:/data/zmaplab#
```

抓包分析——Zmap

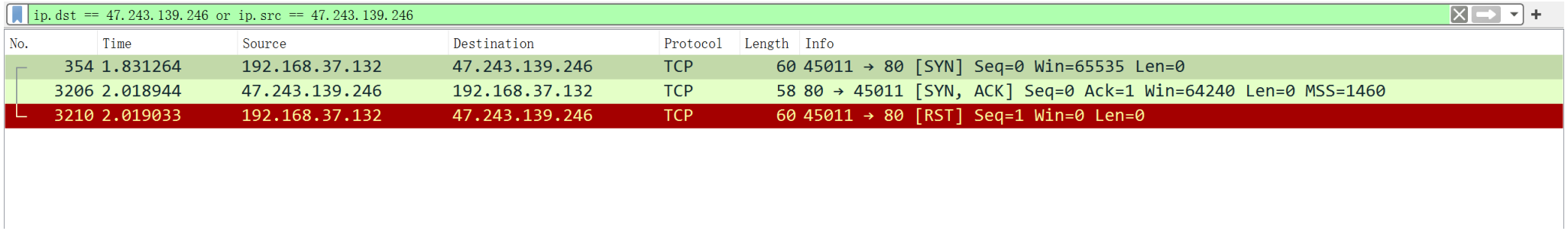
较为保守的情况下，对 10,000 个随机的地址的 80 端口以 10Mbps 的速度扫描。

--bandwidth=10M　　--target-port=80　　--max-targets=10000

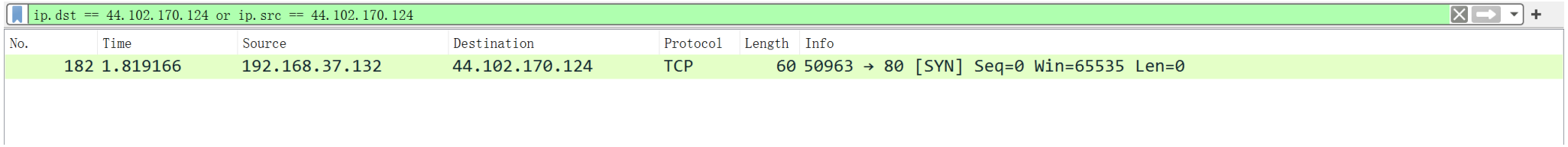
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	VMware_c0:00:08	Broadcast	ARP	42	Who has 192.168.37.2? Tell 192.168.37.1
2	0.934980	VMware_c0:00:08	Broadcast	ARP	42	Who has 192.168.37.2? Tell 192.168.37.1
3	1.806743	192.168.37.132	155.118.7.2	TCP	60	33956 → 80 [SYN] Seq=0 Win=65535 Len=0
4	1.806773	192.168.37.132	60.109.13.78	TCP	60	39090 → 80 [SYN] Seq=0 Win=65535 Len=0
5	1.806898	192.168.37.132	80.16.167.120	TCP	60	55584 → 80 [SYN] Seq=0 Win=65535 Len=0
6	1.807035	192.168.37.132	53.217.231.33	TCP	60	59356 → 80 [SYN] Seq=0 Win=65535 Len=0
7	1.807109	192.168.37.132	110.159.254.162	TCP	60	42829 → 80 [SYN] Seq=0 Win=65535 Len=0
8	1.807192	192.168.37.132	183.26.53.31	TCP	60	59876 → 80 [SYN] Seq=0 Win=65535 Len=0
9	1.807292	192.168.37.132	34.51.235.28	TCP	60	39166 → 80 [SYN] Seq=0 Win=65535 Len=0
10	1.807340	192.168.37.132	86.144.208.218	TCP	60	42631 → 80 [SYN] Seq=0 Win=65535 Len=0
11	1.807430	192.168.37.132	79.15.91.255	TCP	60	54952 → 80 [SYN] Seq=0 Win=65535 Len=0
12	1.807479	192.168.37.132	48.212.79.87	TCP	60	35571 → 80 [SYN] Seq=0 Win=65535 Len=0
13	1.807606	192.168.37.132	202.170.216.217	TCP	60	44931 → 80 [SYN] Seq=0 Win=65535 Len=0
14	1.807625	192.168.37.132	78.46.158.160	TCP	60	60600 → 80 [SYN] Seq=0 Win=65535 Len=0
15	1.807719	192.168.37.132	157.51.120.217	TCP	60	44782 → 80 [SYN] Seq=0 Win=65535 Len=0
16	1.807758	192.168.37.132	189.254.184.88	TCP	60	45638 → 80 [SYN] Seq=0 Win=65535 Len=0
17	1.807850	192.168.37.132	76.216.194.165	TCP	60	57840 → 80 [SYN] Seq=0 Win=65535 Len=0
18	1.807941	192.168.37.132	103.169.91.159	TCP	60	57537 → 80 [SYN] Seq=0 Win=65535 Len=0
19	1.807977	192.168.37.132	48.130.67.79	TCP	60	40478 → 80 [SYN] Seq=0 Win=65535 Len=0
20	1.808108	192.168.37.132	192.16.238.36	TCP	60	55585 → 80 [SYN] Seq=0 Win=65535 Len=0
21	1.808128	192.168.37.132	48.67.37.44	TCP	60	36104 → 80 [SYN] Seq=0 Win=65535 Len=0
22	1.808232	192.168.37.132	143.214.49.152	TCP	60	54686 → 80 [SYN] Seq=0 Win=65535 Len=0

抓包分析——Zmap

1. 向 47.243.139.246 的 80 端口发送 SYN 数据包
2. 接收到 47.243.139.246 的 80 端口的 SYN/ACK 包，证明该 IP 的 80 端口可用
3. 向 47.243.139.246 的 80 端口发送 RST 数据包，防止占用对方资源



Zmap 向其发送 SYN 请求后没有得到应答，故判断该 IP 的 80 端口不可用。



抓包分析——Zmap

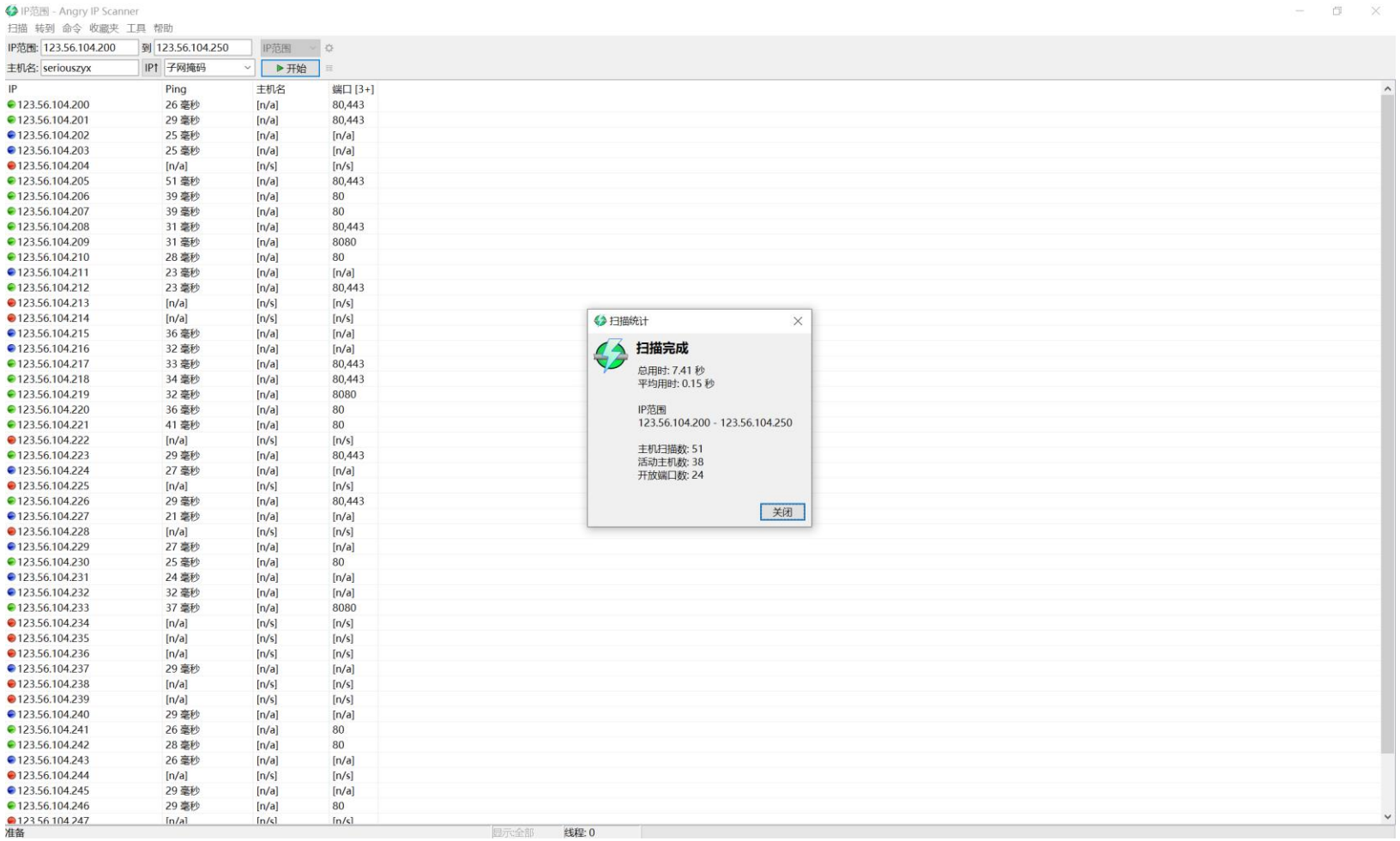
查看 Zmap 向哪些 IP 发送了 RST 数据包，则证明这些 IP 的 80 端口可用。

tcp.flags.reset == 1 and ip.src == 192.168.37.132

No.	Time	Source	Destination		Protocol	Length	Info
3210	2.019033	192.168.37.132	47.243.139.246	1	47.243.139.246		45011 → 80 [RST] Seq=1 Win=0 Len=0
3231	2.020214	192.168.37.132	20.205.204.152	2	20.205.204.152		33597 → 80 [RST] Seq=1 Win=0 Len=0
3232	2.020230	192.168.37.132	121.36.193.65	3	121.36.193.65		46529 → 80 [RST] Seq=1 Win=0 Len=0
3233	2.020232	192.168.37.132	156.245.39.71	4	156.245.39.71		50120 → 80 [RST] Seq=1 Win=0 Len=0
3454	2.034824	192.168.37.132	13.238.233.150	5	13.238.233.150		58325 → 80 [RST] Seq=1 Win=0 Len=0
4257	2.088481	192.168.37.132	142.234.31.240	6	142.234.31.240		34114 → 80 [RST] Seq=1 Win=0 Len=0
5323	2.161347	192.168.37.132	68.183.75.244	7	68.183.75.244		51202 → 80 [RST] Seq=1 Win=0 Len=0
5333	2.161906	192.168.37.132	185.48.122.237	8	185.48.122.237		33463 → 80 [RST] Seq=1 Win=0 Len=0
5360	2.163593	192.168.37.132	52.25.116.123	9	52.25.116.123		37855 → 80 [RST] Seq=1 Win=0 Len=0
5361	2.163601	192.168.37.132	104.127.1.181	10	104.127.1.181		60272 → 80 [RST] Seq=1 Win=0 Len=0
5590	2.179268	192.168.37.132	185.248.102.245	11	185.248.102.245		55986 → 80 [RST] Seq=1 Win=0 Len=0
5947	2.202075	192.168.37.132	95.217.201.8	12	95.217.201.8		58522 → 80 [RST] Seq=1 Win=0 Len=0
6278	2.223394	192.168.37.132	3.125.24.134	13	3.125.24.134		37613 → 80 [RST] Seq=1 Win=0 Len=0
6653	2.247685	192.168.37.132	23.15.117.202	14	23.15.117.202		43651 → 80 [RST] Seq=1 Win=0 Len=0

抓包分析——Angry IP Scanner

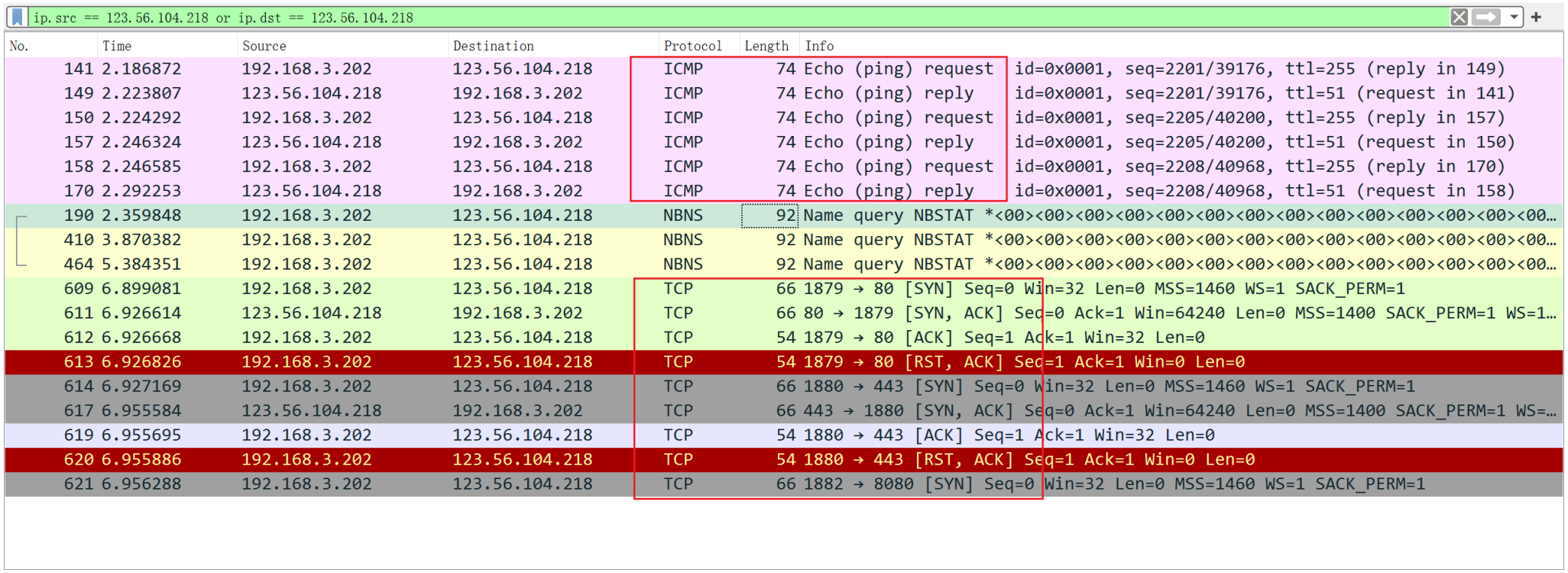
Angry IP Scanner（简称 angryip） 是一款开源跨平台的网络扫描器，主要用于扫描 IP 地址和端口。



- IP 不可用
- IP 可用端口不可用
- IP 和端口均可用

抓包分析——Angry IP Scanner

- angryip 与 123.56.104.218 进行了 3 次 ping，且都予以回复，说明该 IP 可用。
- angryip 分别测试 123.56.104.218 的 80、443 和 8080 端口，其中 80 和 443 端口予以回复，说明这两个端口可用。



ip.src == 123.56.104.218 or ip.dst == 123.56.104.218

No.	Time	Source	Destination	Protocol	Length	Info
141	2.186872	192.168.3.202	123.56.104.218	ICMP	74	Echo (ping) request id=0x0001, seq=2201/39176, ttl=255 (reply in 149)
149	2.223807	123.56.104.218	192.168.3.202	ICMP	74	Echo (ping) reply id=0x0001, seq=2201/39176, ttl=51 (request in 141)
150	2.224292	192.168.3.202	123.56.104.218	ICMP	74	Echo (ping) request id=0x0001, seq=2205/40200, ttl=255 (reply in 157)
157	2.246324	123.56.104.218	192.168.3.202	ICMP	74	Echo (ping) reply id=0x0001, seq=2205/40200, ttl=51 (request in 150)
158	2.246585	192.168.3.202	123.56.104.218	ICMP	74	Echo (ping) request id=0x0001, seq=2208/40968, ttl=255 (reply in 170)
170	2.292253	123.56.104.218	192.168.3.202	ICMP	74	Echo (ping) reply id=0x0001, seq=2208/40968, ttl=51 (request in 158)
190	2.359848	192.168.3.202	123.56.104.218	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00>…
410	3.870382	192.168.3.202	123.56.104.218	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00>…
464	5.384351	192.168.3.202	123.56.104.218	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00>…
609	6.899081	192.168.3.202	123.56.104.218	TCP	66	1879 → 80 [SYN] Seq=0 Win=32 Len=0 MSS=1460 WS=1 SACK_PERM=1
611	6.926614	123.56.104.218	192.168.3.202	TCP	66	80 → 1879 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1400 SACK_PERM=1 WS=1…
612	6.926668	192.168.3.202	123.56.104.218	TCP	54	1879 → 80 [ACK] Seq=1 Ack=1 Win=32 Len=0
613	6.926826	192.168.3.202	123.56.104.218	TCP	54	1879 → 80 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
614	6.927169	192.168.3.202	123.56.104.218	TCP	66	1880 → 443 [SYN] Seq=0 Win=32 Len=0 MSS=1460 WS=1 SACK_PERM=1
617	6.955584	123.56.104.218	192.168.3.202	TCP	66	443 → 1880 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1400 SACK_PERM=1 WS=…
619	6.955695	192.168.3.202	123.56.104.218	TCP	54	1880 → 443 [ACK] Seq=1 Ack=1 Win=32 Len=0
620	6.955886	192.168.3.202	123.56.104.218	TCP	54	1880 → 443 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
621	6.956288	192.168.3.202	123.56.104.218	TCP	66	1882 → 8080 [SYN] Seq=0 Win=32 Len=0 MSS=1460 WS=1 SACK_PERM=1

抓包分析——Angry IP Scanner

angryip 向其发送 3 次 ping 请求，都没有得到回复，则判断其 IP 不可用，也没有向其端口发送数据包。

ip.src == 123.56.104.204 or ip.dst == 123.56.104.204

No.	Time	Source	Destination	Protocol	Length	Info
31	1.754116	192.168.3.202	123.56.104.204	ICMP	74	Echo (ping) request id=0x0001, seq=2167/30472, ttl=255 (no response found…
400	3.499654	192.168.3.202	123.56.104.204	ICMP	74	Echo (ping) request id=0x0001, seq=2283/60168, ttl=255 (no response found…
469	5.495003	192.168.3.202	123.56.104.204	ICMP	74	Echo (ping) request id=0x0001, seq=2296/63496, ttl=255 (no response found…

抓包分析——Masscan

Masscan是一个网络端口扫描工具，它可以在5分钟内扫描整个互联网，从一台机器每秒传输1000万个数据包。
Masscan 默认使用 SYN 扫描，以 IP 123.56.104.218 为例，扫描其 1~600 端口。

```
root@ubuntu:/data# masscan 123.56.104.218 -p1-600

Starting masscan 1.0.3 (http://bit.ly/14GZzcT) at 2021-08-22 13:45:49 GMT
 -- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 1 hosts [600 ports/host]
root@ubuntu:/data#
```

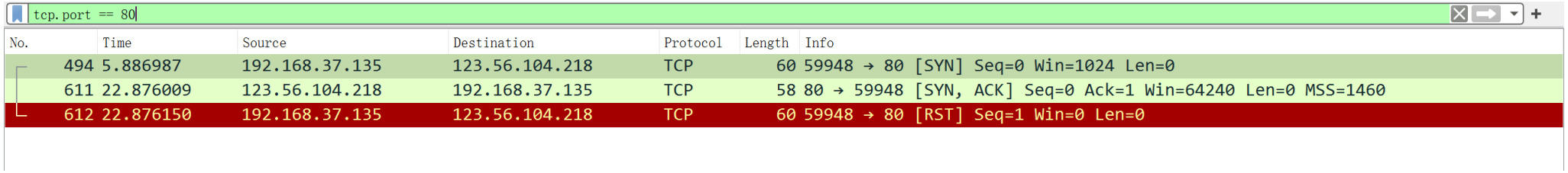
抓包分析——Masscan

Masscan 向 123.56.104.218 的 1~600 端口进行随机化扫描，发出 SYN 请求。

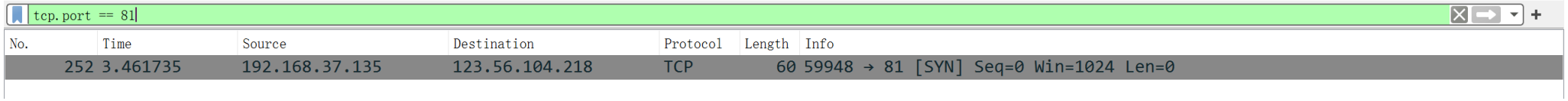
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	VMware_9f:09:8a	Broadcast	ARP	60	Who has 192.168.37.2? Tell 192.168.37.135
2	0.000036	VMware_fc:59:ea	VMware_9f:09:8a	ARP	42	192.168.37.2 is at 00:50:56:fc:59:ea
3	1.002506	192.168.37.135	123.56.104.218	TCP	60	59948 → 522 [SYN] Seq=0 Win=1024 Len=0
4	1.103598	192.168.37.135	123.56.104.218	TCP	60	59948 → 361 [SYN] Seq=0 Win=1024 Len=0
5	1.103629	192.168.37.135	123.56.104.218	TCP	60	59948 → 438 [SYN] Seq=0 Win=1024 Len=0
6	1.103643	192.168.37.135	123.56.104.218	TCP	60	59948 → 413 [SYN] Seq=0 Win=1024 Len=0
7	1.103656	192.168.37.135	123.56.104.218	TCP	60	59948 → 514 [SYN] Seq=0 Win=1024 Len=0
8	1.103678	192.168.37.135	123.56.104.218	TCP	60	59948 → 406 [SYN] Seq=0 Win=1024 Len=0
9	1.103691	192.168.37.135	123.56.104.218	TCP	60	59948 → 390 [SYN] Seq=0 Win=1024 Len=0
10	1.103703	192.168.37.135	123.56.104.218	TCP	60	59948 → 242 [SYN] Seq=0 Win=1024 Len=0
11	1.103722	192.168.37.135	123.56.104.218	TCP	60	59948 → 18 [SYN] Seq=0 Win=1024 Len=0
12	1.103734	192.168.37.135	123.56.104.218	TCP	60	59948 → 121 [SYN] Seq=0 Win=1024 Len=0
13	1.103746	192.168.37.135	123.56.104.218	TCP	60	59948 → 344 [SYN] Seq=0 Win=1024 Len=0
14	1.113399	192.168.37.135	123.56.104.218	TCP	60	59948 → 124 [SYN] Seq=0 Win=1024 Len=0
15	1.123309	192.168.37.135	123.56.104.218	TCP	60	59948 → 547 [SYN] Seq=0 Win=1024 Len=0
16	1.133355	192.168.37.135	123.56.104.218	TCP	60	59948 → 412 [SYN] Seq=0 Win=1024 Len=0
17	1.143066	192.168.37.135	123.56.104.218	TCP	60	59948 → 335 [SYN] Seq=0 Win=1024 Len=0
18	1.153091	192.168.37.135	123.56.104.218	TCP	60	59948 → 424 [SYN] Seq=0 Win=1024 Len=0
19	1.162534	192.168.37.135	123.56.104.218	TCP	60	59948 → 295 [SYN] Seq=0 Win=1024 Len=0
20	1.172531	192.168.37.135	123.56.104.218	TCP	60	59948 → 325 [SYN] Seq=0 Win=1024 Len=0
21	1.182560	192.168.37.135	123.56.104.218	TCP	60	59948 → 373 [SYN] Seq=0 Win=1024 Len=0
22	1.192576	192.168.37.135	123.56.104.218	TCP	60	59948 → 298 [SYN] Seq=0 Win=1024 Len=0
23	1.202943	192.168.37.135	123.56.104.218	TCP	60	59948 → 508 [SYN] Seq=0 Win=1024 Len=0
24	1.213332	192.168.37.135	123.56.104.218	TCP	60	59948 → 349 [SYN] Seq=0 Win=1024 Len=0
25	1.222653	192.168.37.135	123.56.104.218	TCP	60	59948 → 225 [SYN] Seq=0 Win=1024 Len=0
26	1.232859	192.168.37.135	123.56.104.218	TCP	60	59948 → 77 [SYN] Seq=0 Win=1024 Len=0
27	1.242562	192.168.37.135	123.56.104.218	TCP	60	59948 → 75 [SYN] Seq=0 Win=1024 Len=0
28	1.252685	192.168.37.135	123.56.104.218	TCP	60	59948 → 448 [SYN] Seq=0 Win=1024 Len=0
29	1.262767	192.168.37.135	123.56.104.218	TCP	60	59948 → 451 [SYN] Seq=0 Win=1024 Len=0
30	1.272970	192.168.37.135	123.56.104.218	TCP	60	59948 → 379 [SYN] Seq=0 Win=1024 Len=0

抓包分析——Masscan

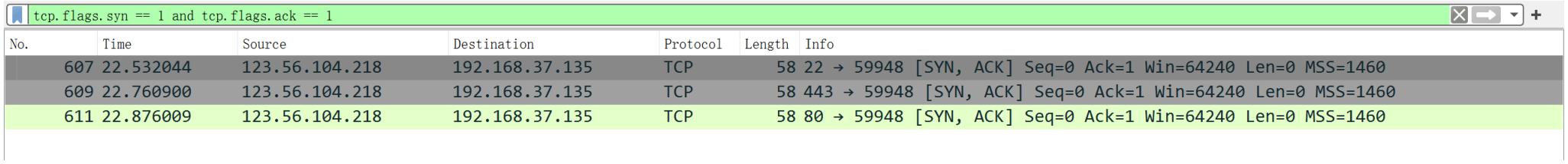
查看 80 端口的数据包，下图可知 80 端口向 Masscan 回复，说明该端口可用。



查看 81 端口的数据包，发现并没有数据包回复，说明该端口不可用。



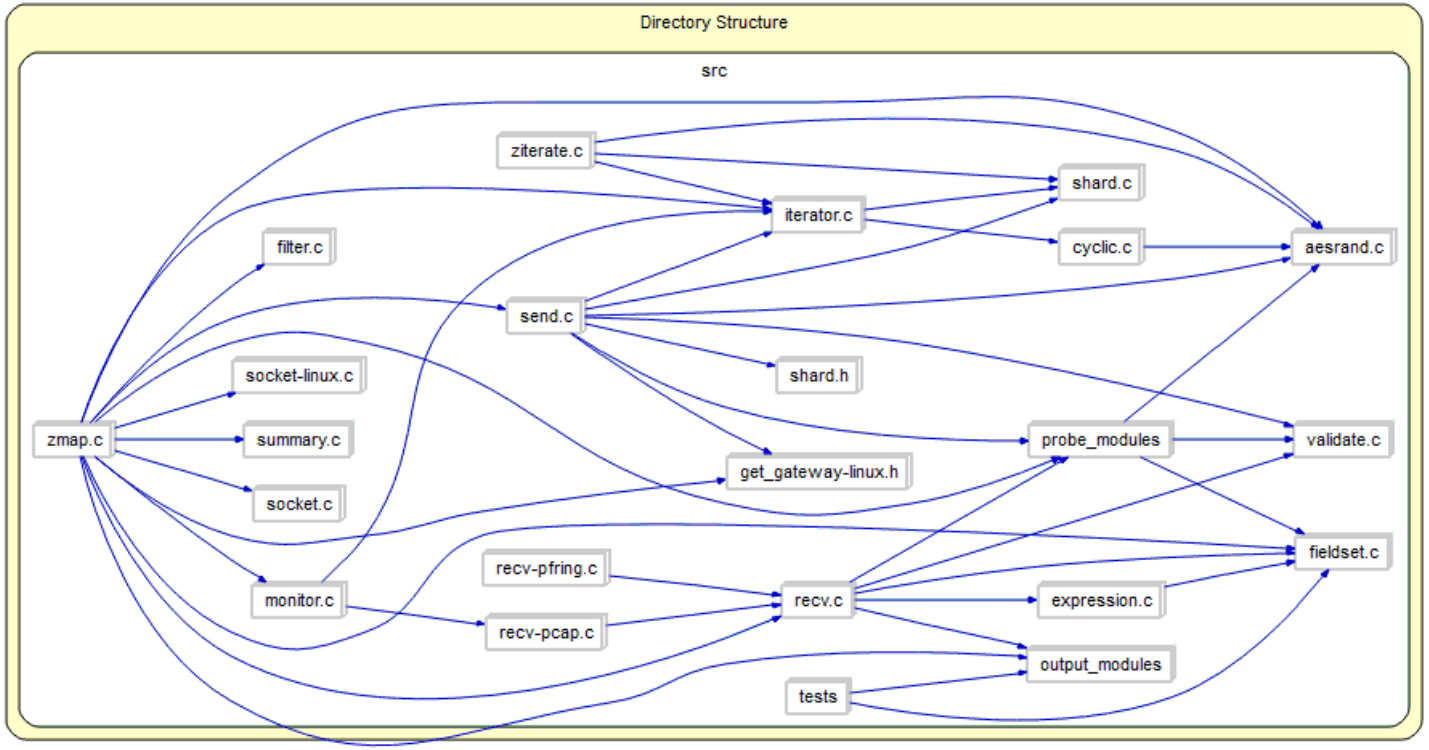
筛选收到的 SYN/ACK 数据包，得到 22、443 和 80 端口，说明 123.56.104.218 的 1~600 中这 3 个端口可用。



源码分析

源码分析——Zmap

通过Zmap整体函数调用图可以直观的看到整个程序调用的过程。
Zmap 在启动时候，先获取环境信息，如 IP、网关等。然后读取配置文件选择使用哪种扫描方式，然后在 Probe_modules 切换到对应的模块，然后启动。



源码分析——Zmap

zmap/src/probe_modules/module_tcp_synscan.c 是用于执行 TCP SYN 扫描的探测模块，在初始化阶段的 synscan_init_perthread 函数中，依次调用 make_ip_header 函数和 make_tcp_header 函数进行数据包 header 的封装。

```
1  static int synscan_init_perthread(
2      void *buf, macaddr_t *src, macaddr_t *gw,
3      port_h_t dst_port,
4      UNUSED void **arg_ptr)
5  {
6      struct ether_header *eth_header = (struct ether_header *)buf;
7      make_eth_header(eth_header, src, gw);
8      struct ip *ip_header = (struct ip *)(&eth_header[1]);
9      uint16_t len = htons(sizeof(struct ip) + ZMAP_TCP_SYNSCAN_TCP_HEADER_LEN);
10     make_ip_header(ip_header, IPPROTO_TCP, len);
11     struct tcphdr *tcp_header = (struct tcphdr *)(&ip_header[1]);
12     make_tcp_header(tcp_header, dst_port, TH_SYN);
13     set_mss_option(tcp_header);
14     return EXIT_SUCCESS;
15 }
```

源码分析——Zmap

这两个函数编写于 zmap/src/probe_modules/packet.c 中。分析 make_ip_header 函数可知，在下示第 7 行，IP 的 identification number 被设置为固定的 54321。

```
1  void make_ip_header(struct ip *iph, uint8_t protocol, uint16_t len)
2  {
3      iph->ip_hl = 5;  // Internet Header Length
4      iph->ip_v = 4;   // IPv4
5      iph->ip_tos = 0; // Type of Service
6      iph->ip_len = len;
7      iph->ip_id = htons(54321); // identification number
8      iph->ip_off = 0;       // fragmentation flag
9      iph->ip_ttl = MAXTTL;      // time to live (TTL)
10     iph->ip_p = protocol;      // upper layer protocol => TCP
11     // we set the checksum = 0 for now because that's
12     // what it needs to be when we run the IP checksum
13     iph->ip_sum = 0;
14 }
```

源码分析——Zmap

分析 make_tcp_header 函数可知，在下示第 10 行，TCP 的 window 被设置为固定的 65535。

```
1  void make_tcp_header(struct tcphdr *tcp_header, port_h_t dest_port,
2                       uint16_t th_flags)
3  {
4      tcp_header->th_seq = random();
5      tcp_header->th_ack = 0;
6      tcp_header->th_x2 = 0;
7      tcp_header->th_off = 5; // data offset
8      tcp_header->th_flags = 0;
9      tcp_header->th_flags |= th_flags;
10     tcp_header->th_win = htons(65535); // largest possible window
11     tcp_header->th_sum = 0;
12     tcp_header->th_urp = 0;
13     tcp_header->th_dport = htons(dest_port);
14 }
```

源码分析——Zmap

查看抓取的 SYN 数据包，如下图所示，IP 的 ID 和 TCP 的 window 确实为 54321 和 65535，所以这两个固定值可作为扫描器特征。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	VMware_c0:00:08	Broadcast	ARP	42	Who has 192.168.37.2? Tell 192.168.37.1
2	0.934980	VMware_c0:00:08	Broadcast	ARP	42	Who has 192.168.37.2? Tell 192.168.37.1
3	1.806743	192.168.37.132	155.118.7.2	TCP	60	33956 → 80 [SYN] Seq=0 Win=65535 Len=0
4	1.806773	192.168.37.132	60.109.13.78	TCP	60	39090 → 80 [SYN] Seq=0 Win=65535 Len=0

```
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 40
  Identification: 0xd431 (54321)
> Flags: 0x00
  Fragment Offset: 0
  Time to Live: 255
  Protocol: TCP (6)
  Header Checksum: 0x5ef9 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.37.132
  Destination Address: 155.118.7.2
v Transmission Control Protocol, Src Port: 33956, Dst Port: 80, Seq: 0, Len: 0
  Source Port: 33956
  Destination Port: 80
  [Stream index: 0]
  [TCP Segment Len: 0]
  Sequence Number: 0    (relative sequence number)
  Sequence Number (raw): 1424768786
  [Next Sequence Number: 1    (relative sequence number)]
  Acknowledgment Number: 0
  Acknowledgment number (raw): 0
  0101 .... = Header Length: 20 bytes (5)
> Flags: 0x002 (SYN)
  Window: 65535
  [Calculated window size: 65535]
  Checksum: 0x0e4b [unverified]
  [Checksum Status: Unverified]
```

抓包分析——Angry IP Scanner

分析 ipscan/test/net/azib/ipscan/core/net/ICMPSharedPingerTest.java 源码，该测试类调用 pinger.ping()方法 3 次，并计算平均时长。

```
1  public class ICMPSharedPingerTest {
2      @Test @Ignore("this test works only under root")
3      public void testPing() throws Exception {
4          Pinger pinger = new ICMPSharedPinger(1000);
5          PingResult result = pinger.ping(new ScanningSubject(InetAddress.getLocalHost()), 3);
6          assertTrue(result.getAverageTime() >= 0);
7          assertTrue(result.getAverageTime() < 50);
8          assertTrue(result.getTTL() >= 0);
9      }
10 }
```

抓包分析——Angry IP Scanner

该方法在 ipscan/test/net/azib/ipscan/core/net/WindowsPinger.java 中，源码如下所示，判断 IP 类型，并调用 IPv6 和 IPv4 对应的方法。

```
1 public PingResult ping(ScanningSubject subject, int count) throws IOException {
2     if (subject.isIPv6())
3         return ping6(subject, count);
4     else
5         return ping4(subject, count);
6 }
```

抓包分析——Angry IP Scanner

以 IPv4 为例，方法中定义了数据包的数据大小为 32，即 sendDataSize = 32。后续使用 Memory()方法创建 SendData 对象，并未对其进行赋值，故默认值应全为 0。

```
1  private PingResult ping4(ScanningSubject subject, int count) throws IOException {
2      Pointer handle = dll.IcmpCreateFile();
3      if (handle == null) throw new IOException("Unable to create Windows native ICMP handle");
4
5      int sendDataSize = 32;
6      int replyDataSize = sendDataSize + (new IcmpEchoReply().size()) + 10;
7      Pointer sendData = new Memory(sendDataSize);
8      sendData.clear(sendDataSize);
9      Pointer replyData = new Memory(replyDataSize);
10
11     PingResult result = new PingResult(subject.getAddress(), count);
12     try {
13         IpAddrByVal ipaddr = toIpAddr(subject.getAddress());
14         for (int i = 1; i <= count && !currentThread().isInterrupted(); i++) {
15             int numReplies = dll.IcmpSendEcho(handle, ipaddr, sendData, (short) sendDataSize, null, replyData, replyDataSize, timeout);
16             IcmpEchoReply echoReply = new IcmpEchoReply(replyData);
17             if (numReplies > 0 && echoReply.status == 0 && Arrays.equals(echoReply.address.bytes, ipaddr.bytes)) {
18                 result.addReply(echoReply.roundTripTime);
19                 result.setTTL(echoReply.options.ttl & 0xFF);
20             }
21         }
22     }
23     finally {
24         dll.IcmpCloseHandle(handle);
25     }
26     return result;
27 }
```

抓包分析——Angry IP Scanner

在实际抓包中，每个发出的 ICMP 请求中，Data 的大小均为 32 字节，且全为 0，所以可将它作为 angryip 的特征。

No.	Time	Source	Destination	Protocol	Length	Info
10	1.660070	123.56.104.200	192.168.3.202	ICMP	74	Echo (ping) reply id=0x0001, seq=2156/27656, ttl=51 (request in 9)
11	1.660610	192.168.3.202	123.56.104.200	ICMP	74	Echo (ping) request id=0x0001, seq=2157/27912, ttl=255 (reply in 13)
12	1.662938	192.168.3.202	123.56.104.201	ICMP	74	Echo (ping) request id=0x0001, seq=2158/28168, ttl=255 (reply in 16)
13	1.689967	123.56.104.200	192.168.3.202	ICMP	74	Echo (ping) reply id=0x0001, seq=2157/27912, ttl=51 (request in 11)
14	1.690448	192.168.3.202	123.56.104.200	ICMP	74	Echo (ping) request id=0x0001, seq=2159/28424, ttl=255 (reply in 18)
15	1.694586	192.168.3.202	123.56.104.202	ICMP	74	Echo (ping) request id=0x0001, seq=2160/28680, ttl=255 (reply in 20)
16	1.700865	123.56.104.201	192.168.3.202	ICMP	74	Echo (ping) reply id=0x0001, seq=2158/28168, ttl=49 (request in 12)

```
> Frame 12: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
> Ethernet II, Src: IntelCor_1a:b4:86 (f0:77:c3:1a:b4:86), Dst: HuaweiDe_a7:69:16 (e4:26:8b:a7:69:16)
> Internet Protocol Version 4, Src: 192.168.3.202, Dst: 123.56.104.201
v Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xef90 [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence Number (BE): 2158 (0x086e)
    Sequence Number (LE): 28168 (0x6e08)
    [Response frame: 16]
  v Data (32 bytes)
      Data: 00000000000000000000000000000000000000000000000000000000000000000000
      [Length: 32]
```

抓包分析——Masscan

观察抓包分析中结果可以发现，所有发出的 SYN 请求中，窗口大小都是 1024。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	VMware_9f:09:8a	Broadcast	ARP	60	Who has 192.168.37.2? Tell 192.168.37.135
2	0.000036	VMware_fc:59:ea	VMware_9f:09:8a	ARP	42	192.168.37.2 is at 00:50:56:fc:59:ea
3	1.002506	192.168.37.135	123.56.104.218	TCP	60	59948 → 522 [SYN] Seq=0 Win=1024 Len=0
4	1.103598	192.168.37.135	123.56.104.218	TCP	60	59948 → 361 [SYN] Seq=0 Win=1024 Len=0
5	1.103629	192.168.37.135	123.56.104.218	TCP	60	59948 → 438 [SYN] Seq=0 Win=1024 Len=0
6	1.103643	192.168.37.135	123.56.104.218	TCP	60	59948 → 413 [SYN] Seq=0 Win=1024 Len=0
7	1.103656	192.168.37.135	123.56.104.218	TCP	60	59948 → 514 [SYN] Seq=0 Win=1024 Len=0
8	1.103678	192.168.37.135	123.56.104.218	TCP	60	59948 → 406 [SYN] Seq=0 Win=1024 Len=0
9	1.103691	192.168.37.135	123.56.104.218	TCP	60	59948 → 390 [SYN] Seq=0 Win=1024 Len=0
10	1.103703	192.168.37.135	123.56.104.218	TCP	60	59948 → 242 [SYN] Seq=0 Win=1024 Len=0
11	1.103722	192.168.37.135	123.56.104.218	TCP	60	59948 → 18 [SYN] Seq=0 Win=1024 Len=0
12	1.103734	192.168.37.135	123.56.104.218	TCP	60	59948 → 121 [SYN] Seq=0 Win=1024 Len=0
13	1.103746	192.168.37.135	123.56.104.218	TCP	60	59948 → 344 [SYN] Seq=0 Win=1024 Len=0
14	1.113399	192.168.37.135	123.56.104.218	TCP	60	59948 → 124 [SYN] Seq=0 Win=1024 Len=0
15	1.123309	192.168.37.135	123.56.104.218	TCP	60	59948 → 547 [SYN] Seq=0 Win=1024 Len=0
16	1.133355	192.168.37.135	123.56.104.218	TCP	60	59948 → 412 [SYN] Seq=0 Win=1024 Len=0
17	1.143066	192.168.37.135	123.56.104.218	TCP	60	59948 → 335 [SYN] Seq=0 Win=1024 Len=0
18	1.153091	192.168.37.135	123.56.104.218	TCP	60	59948 → 424 [SYN] Seq=0 Win=1024 Len=0
19	1.162534	192.168.37.135	123.56.104.218	TCP	60	59948 → 295 [SYN] Seq=0 Win=1024 Len=0
20	1.172531	192.168.37.135	123.56.104.218	TCP	60	59948 → 325 [SYN] Seq=0 Win=1024 Len=0
21	1.182560	192.168.37.135	123.56.104.218	TCP	60	59948 → 373 [SYN] Seq=0 Win=1024 Len=0
22	1.192576	192.168.37.135	123.56.104.218	TCP	60	59948 → 298 [SYN] Seq=0 Win=1024 Len=0
23	1.202943	192.168.37.135	123.56.104.218	TCP	60	59948 → 508 [SYN] Seq=0 Win=1024 Len=0
24	1.213332	192.168.37.135	123.56.104.218	TCP	60	59948 → 349 [SYN] Seq=0 Win=1024 Len=0
25	1.222653	192.168.37.135	123.56.104.218	TCP	60	59948 → 225 [SYN] Seq=0 Win=1024 Len=0
26	1.232859	192.168.37.135	123.56.104.218	TCP	60	59948 → 77 [SYN] Seq=0 Win=1024 Len=0
27	1.242562	192.168.37.135	123.56.104.218	TCP	60	59948 → 75 [SYN] Seq=0 Win=1024 Len=0
28	1.252685	192.168.37.135	123.56.104.218	TCP	60	59948 → 448 [SYN] Seq=0 Win=1024 Len=0
29	1.262767	192.168.37.135	123.56.104.218	TCP	60	59948 → 451 [SYN] Seq=0 Win=1024 Len=0
30	1.272970	192.168.37.135	123.56.104.218	TCP	60	59948 → 379 [SYN] Seq=0 Win=1024 Len=0

抓包分析——Masscan

在 Masscan 的主函数 masscan/src/main.c 文件中，默认使用以下代码初始化 TCP 数据包的模板。

```
template_packet_init(
    parms->tmplset,
    parms->source_mac,
    parms->router_mac_ipv4,
    parms->router_mac_ipv6,
    masscan->payloads.udp,
    masscan->payloads.oproto,
    stack_if_datalink(masscan->nic[index].adapter),
    masscan->seed);
```

该函数位于 masscan/src/templ.pkt.c 中，其中对于 TCP 的初始化代码如下所示。

```
/* [TCP] */
_template_init(&templset->pkts[Proto_TCP],
               source_mac, router_mac_ipv4, router_mac_ipv6,
               default_tcp_template,
               sizeof(default_tcp_template)-1,
               data_link);
templset->count++;
```

抓包分析——Masscan

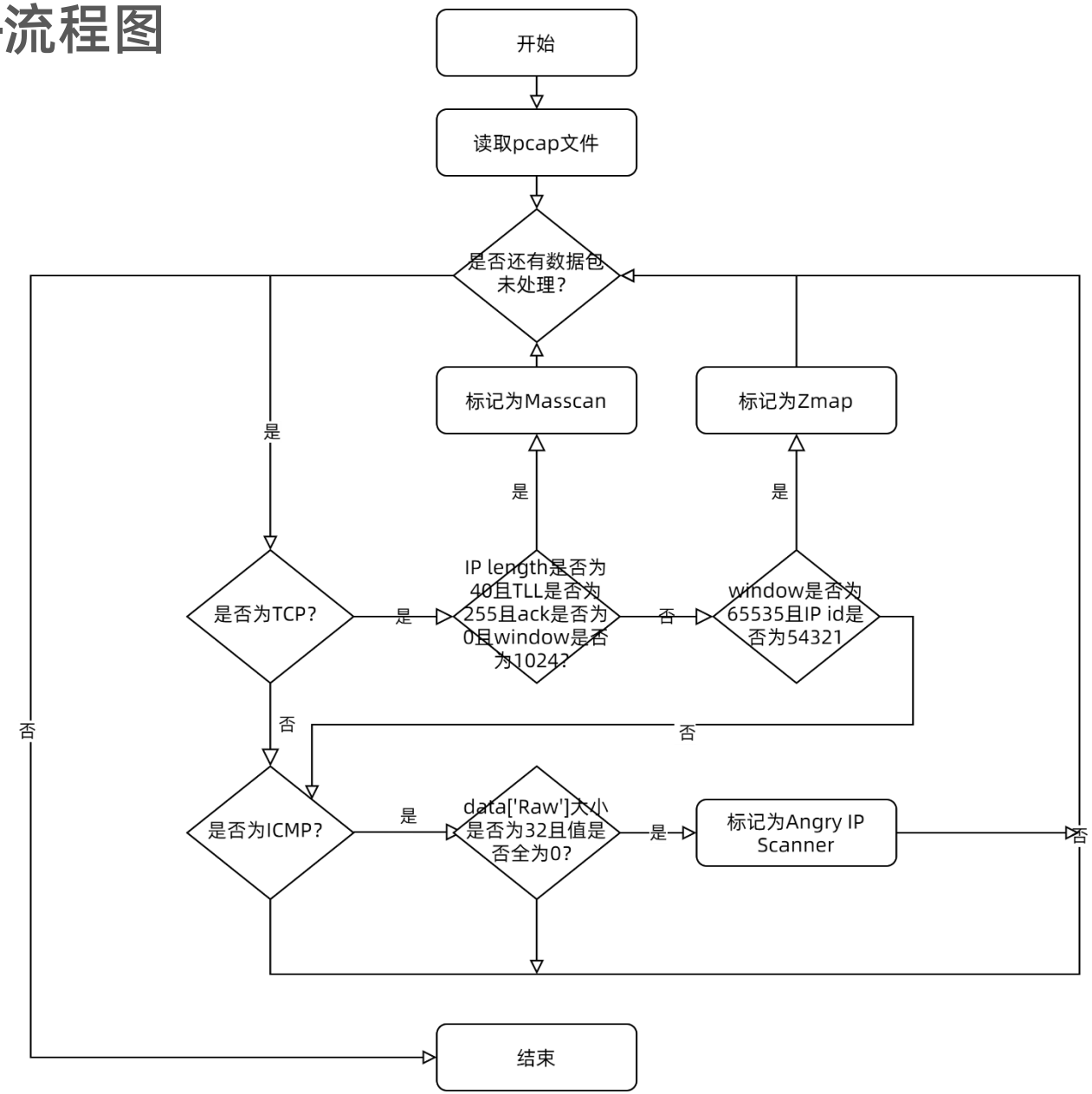
其中调用的 default_tcp_template 定义在该文件头部，下述 7 行指定 IP 的 length 为 40，下述 10 行指定 TLL 为 255，下述 18 行指定 ack 为 0，下述 21 行指定 window 的大小为固定的 1024，所以可以将其视为 Masscan 的特征。

```
1  static unsigned char default_tcp_template[] =
2      "\0\1\2\3\4\5"  /* Ethernet: destination */
3      "\6\7\x8\x9\xa\xb"  /* Ethernet: source */
4      "\x08\x00"      /* Ethernet type: IPv4 */
5      "\x45"          /* IP type */
6      "\x00"
7      "\x00\x28"      /* total length = 40 bytes */
8      "\x00\x00"      /* identification */
9      "\x00\x00"      /* fragmentation flags */
10     "\xFF\x06"      /* TTL=255, proto=TCP */
11     "\xFF\xFF"      /* checksum */
12     "\0\0\0\0"      /* source address */
13     "\0\0\0\0"      /* destination address */
14
15     "\0\0"          /* source port */
16     "\0\0"          /* destination port */
17     "\0\0\0\0"      /* sequence number */
18     "\0\0\0\0"      /* ack number */
19     "\x50"          /* header length */
20     "\x02"          /* SYN */
21     "\x04\x0"        /* window fixed to 1024 */
22     "\xFF\xFF"      /* checksum */
23     "\x00\x00"      /* urgent pointer */
24     "\x02\x04\x05\xb4"  /* added options [mss 1460] */
25 ;
```

扫描探测工具	特征
Zmap	IP ID：54321 Window：65535
Angry IP Scanner	ICMP Data：32bytes，全为0
X-Scan -> Masscan	IP len：40 TTL：255 ack：0 Window：1024

Demo实现

Demo实现——流程图


开始
读取pcap文件
是否还有数据包未处理?
标记为Masscan
标记为Zmap
是
是
是
是否为TCP?
是
IP length是否为40且TLL是否为255且ack是否为0且window是否为1024?
否
window是否为65535且IP id是否为54321
否
否
否
是否为ICMP?
是
data['Raw']大小是否为32且值是否全为0?
是
标记为Angry IP Scanner
否
结束

Demo实现——代码

```
for data in packets:
    if 'TCP' in data:
        # 识别 Zmap
        if (data['TCP'].window == 65535) and (data['IP'].id == 54321):
            isZmap = True
        # 识别 Masscan
        if data['TCP'].window == 1024 and data['TCP'].ack == 0 \
                and data['IP'].ttl == 255 and data['IP'].len == 40:
            isMasscan = True
    # 识别 Angry IP Scanner
    if 'ICMP' in data:
        if 'Raw' in data:
            items = processStr(data['Raw'].load)
            if len(data['Raw']) == 32 and items == ANGRYIP_FLAG:
                isAngryip = True
```

小组总结

小组总结——项目时间线

作业分配

腾讯会议讨论各自基本情况及作业任务的分配

搭建环境

搭建环境安装Nmap，搜索相关资料，研究功能及源码

探究思路

细化需求，搜索相关解决思路

分析Zmap、Angryip

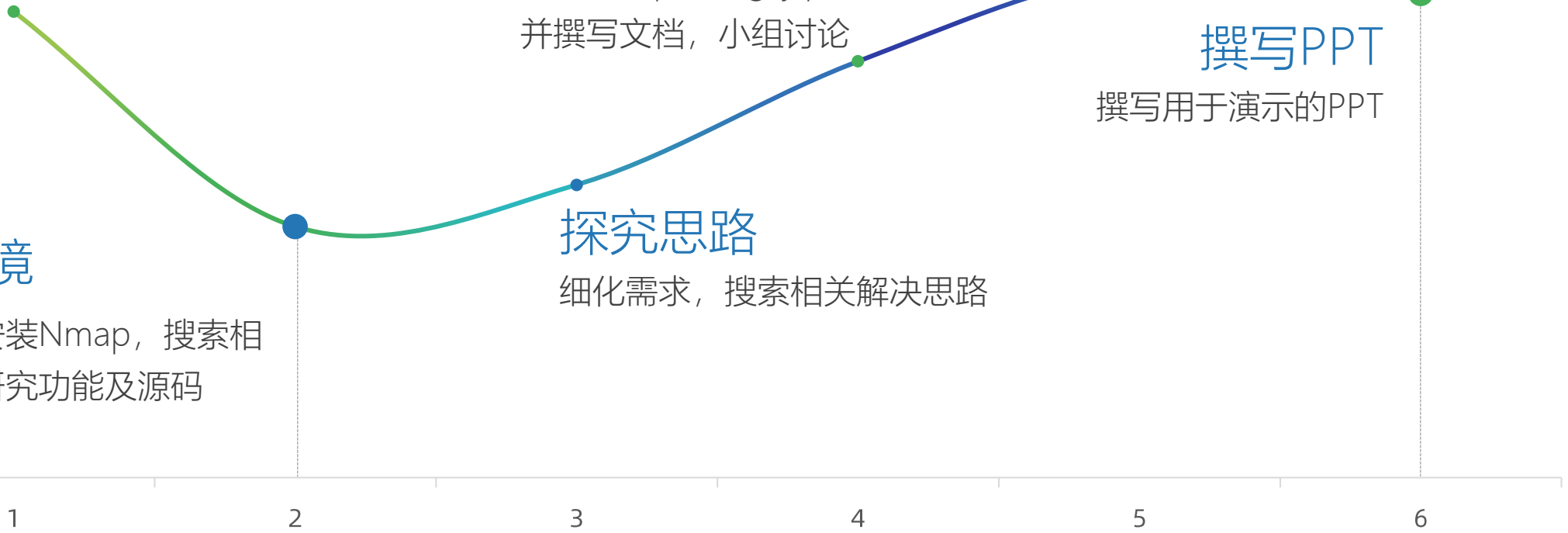
从抓包分析和源码分析两个方面对Zmap和Angryip进行分析并撰写文档，小组讨论

编写代码

编写扫描器工具识别Demo，并反应X-Scan的问题，补充Masscan的分析及代码

撰写PPT

撰写用于演示的PPT



谢谢观赏

《扫描探测工具识别》7组

汇报人：赵奕翔
小组成员：畅晨铭
田　婕
导师：周　舟
时间：2021年8月25日