北京邮电大学

网络空间安全学院



**软件详细设计报告**

**项目： 基于DNS流量分析的僵尸网络检测工具**

**组员： 王硕、彭致远、李懿飞、王晨旭**

**2020年12月4日**

目录

[1 引言 4](#_Toc58503450)

[1.1 目的 4](#_Toc58503451)

[1.2 背景及范围 4](#_Toc58503452)

[1.3 定义、术语或缩略语 4](#_Toc58503453)

[2 软件系统结构 5](#_Toc58503454)

[2.1 需求概述 5](#_Toc58503455)

[2.2 子模块划分 5](#_Toc58503456)

[2.3 子模块关系 6](#_Toc58503457)

[3 各模块详细设计 8](#_Toc58503458)

[3.1 DNS数据包捕获模块 8](#_Toc58503459)

[3.1.1 模块功能描述 8](#_Toc58503460)

[3.1.2 利用DNS隧道工具捕获恶意数据包 8](#_Toc58503461)

[3.1.3 实时检测中的DNS流量获取 11](#_Toc58503462)

[3.1.3.1 思路 11](#_Toc58503463)

[3.1.3.2 算法 11](#_Toc58503464)

[3.1.4 输入输出与性能 12](#_Toc58503465)

[3.2 PCAP文件解析模块 13](#_Toc58503466)

[3.2.1 模块功能描述 13](#_Toc58503467)

[3.2.2 PcapParser类 14](#_Toc58503468)

[3.2.3 输入 21](#_Toc58503469)

[3.2.4 输出 21](#_Toc58503470)

[3.3 DNS指纹生成模块 23](#_Toc58503471)

[3.3.1 模块功能描述 23](#_Toc58503472)

[3.3.2 DNS特征提取 23](#_Toc58503473)

[3.3.3 DNS特征解释 27](#_Toc58503474)

[3.3.4 模块输出 34](#_Toc58503475)

[3.4 机器学习特征分析模块 35](#_Toc58503476)

[3.4.1 模块功能描述 35](#_Toc58503477)

[3.4.2 输入 35](#_Toc58503478)

[3.4.3 算法 36](#_Toc58503479)

[3.4.4 特征重要度分析 42](#_Toc58503480)

[3.4.5 模型调优 42](#_Toc58503481)

[3.4.6 准确率 43](#_Toc58503482)

[3.4.6.1 混淆矩阵 43](#_Toc58503483)

[3.4.6.2 准确率 43](#_Toc58503484)

[3.4.7 模型保存 44](#_Toc58503485)

[3.5 机器学习分类模块 45](#_Toc58503486)

[3.5.1 模块功能描述 45](#_Toc58503487)

[3.5.2 算法 45](#_Toc58503488)

[3.5.3 输出 46](#_Toc58503489)

[3.6 辅助输出模块 47](#_Toc58503490)

[3.6.1 模块功能描述 47](#_Toc58503491)

[3.6.2 算法 47](#_Toc58503492)

[3.6.3 输入输出 53](#_Toc58503493)

# 1 引言

### 1.1 目的

编写该详细设计说明书是为说明该项目——基于DNS流量分析的僵尸网络检测工具的设计考虑，包括程序描述、输入/输出、算法和流程逻辑等，为软件编程和系统维护提供基础。本说明书的预期读者为系统设计人员、软件开发人员、软件测试人员和项目评审人员。其中系统设计人员、软件开发人员、软件测试人员为小组内部成员组成，软件评审人员为课程老师或助教。

### 1.2 背景及范围

* + 项目名称：基于DNS流量分析的僵尸网络检测工具
  + 项目成员：北京邮电大学网络空间安全学院“网络安全分析实践”课程开发小组
    - 王硕（组长）：2018213641
    - 彭致远：2018213646
    - 李懿飞：2018213632
    - 王晨旭：2018213636
  + 系统范围：具有python2.7.9环境的Windows系统计算机
  + 用户：无限制
  + 实现项目的计算机网络：校园网

### 1.3 定义、术语或缩略语

|  |  |  |
| --- | --- | --- |
| 序号 | 术语或缩写 | 解释 |
| 1 | DNS | 域名系统服务协议，是一种分布式网络目录服务，主要用于域名与 IP 地址的相互转换，以及控制因特网的电子邮件的发送 |
| 2 | DNS特征 | 用来衡量主机进行DNS通信过程的行为 |
| 3 | DNS指纹 | 根据特征工程，提取主机DNS通信特征，对主机行为进行了多维度标识 |
| 4 | Botnet | 僵尸网络，是指采用一种或多种传播手段，将大量主机感染bot程序（僵尸程序）病毒，从而在控制者和被感染主机之间所形成的一个可一对多控制的网络 |
| 5 | 僵尸主机 | 本文表示处于僵尸网络中的主机 |
| 6 | 域/域名 | 域名（英语：Domain Name），是由一串用点分隔的名字组成的Internet上某一台计算机或计算机组的名称，用于在数据传输时对计算机的定位标识 |
| 7 | MX记录 | 邮件交换记录 (MX record)是域名系统（DNS）中的一种资源记录类型，用于指定负责处理发往收件人域名的邮件服务器。 |
| 8 | PTR记录 | PTR记录，是电子邮件系统中的邮件交换记录的一种；另一种邮件交换记录是A记录（在IPv4协议中）或AAAA记录（在IPv6协议中）。PTR记录常被用于反向地址解析。 |
| 9 | TLD | top level domain，顶级域名 |
| 10 | SLD | second level domain，二级域名 |
| 11 | DGA域名 | dga是一种算法，作用生成随机数的。用dga算法生成的域名，这种域名通常硬编码在恶意软件中。 |

# 2 软件系统结构

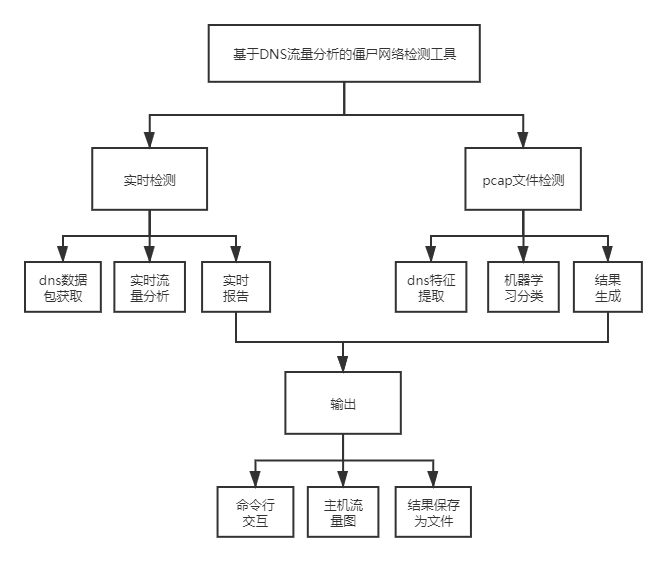
### 2.1 需求概述

开发一个基于DNS流量分析的僵尸网络检测工具，能够通过网络中的数据流量，检测出一个网络中的僵尸主机。工具应具有文件检测和实时检测的功能，并提供友好的用户交互与完善的输入输出系统。

该工具应能够部署在网络边界上，有效地检测通过DNS隐蔽信道传递信息的主机，区分出正常主机与感染了病毒的僵尸主机，检测准确率较现有的检测工具高，误报、漏报率低，时长尽可能短，能够记录僵尸主机的活动、行为，留下检测记录。

### 2.2 子模块划分

本工具的系统功能图如下所示：

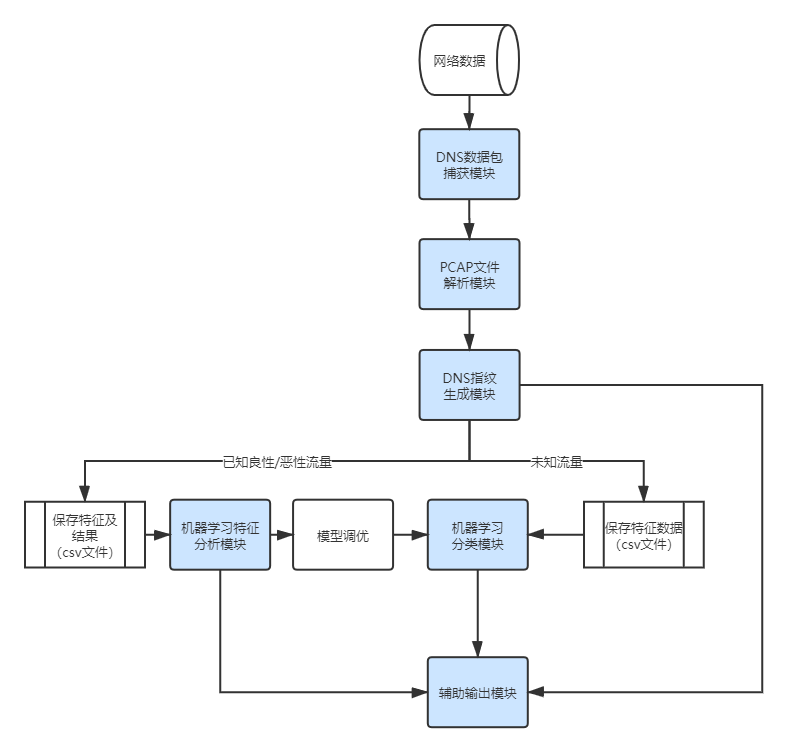


根据系统功能图，可以将本项目划分为六个模块，分别为：

1. DNS数据包捕获模块
2. PCAP文件解析模块
3. DNS指纹生成模块
4. 机器学习特征分析模块
5. 机器学习分类模块
6. 辅助输出模块

### 2.3 子模块关系

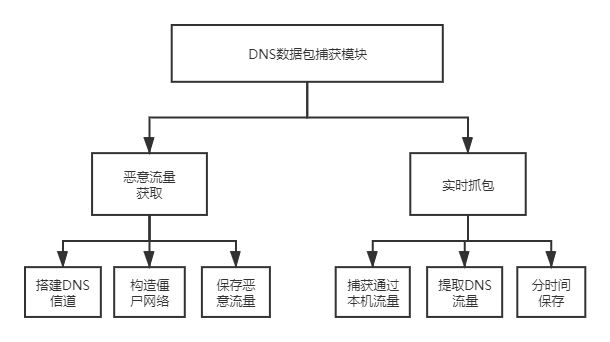
以模块化的方式展示整个系统的流程图如下所示：



# 3 各模块详细设计

### 3.1 DNS数据包捕获模块

#### 3.1.1 模块功能描述



本模块的功能为获取DNS数据包。

一方面用于机器学习，包括获取正常的数据包和恶意的数据包。正常的数据包可以从公开流量数据集中获取，异常的数据包考虑构建一套DNS数据制造和收集的自动化框架，涵盖几种常见的DNS隧道工具（iodine/ozymandns/dns2tcp/dnscat2/Cobalt Strike）。

另一方面用于实时检测中的DNS数据包的抓包、过滤、切片。

#### 3.1.2 利用DNS隧道工具捕获恶意数据包

实现DNS隧道的工具有很多，不同工具在工作原理上相似，差异在于其通信方式、编码加密类型等。为了使机器学习具备足够全面和大量的训练样本，准备构建了一套DNS数据制造和收集的自动化框架。目前市场上有的一些DNS隧道工具有以下几个：

* Iodine
* Ozymandns
* Dns2tcp
* Dnscat2
* Cobalt Strike

这里以工具Iodine为例说明其原理配置过程：

1. 原理：

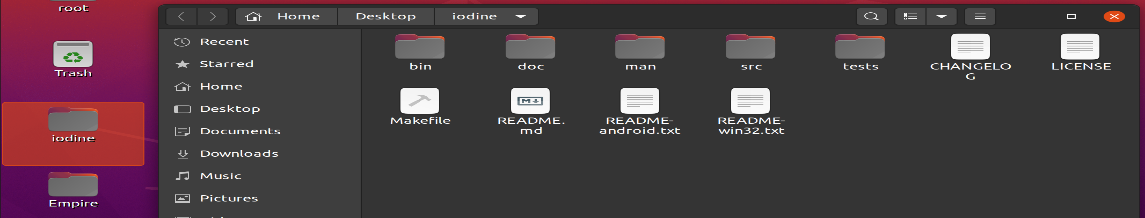
iodine是一个流行的DNS隧道工具，它将IPv4数据封装到DNS协议中传输，安装部署可以很方便的通过yum或apt-get完成，也可以自行编译安装。

因为 iodine（碘）的原子序数为53，这恰好是DNS端口号，故取名为iodine。

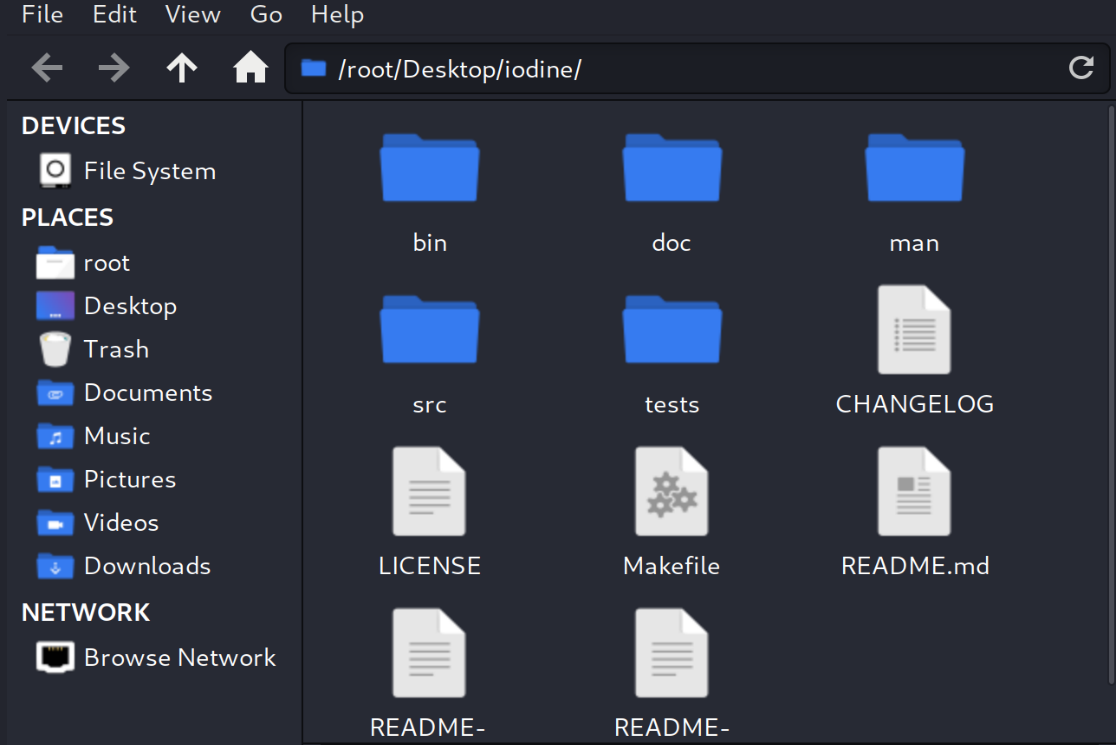
iodine基于C语言开发，分为服务端程序 iodined 和客户端程序 iodine，kali系统已内置。服务端作为攻击机，客户端作为被控机。环境搭建完成后，由于服务端和客户端会处于同一个局域网，因此两端均可任意放于主控机和被控机上（具体还是得视情况而定）

1. 配置过程：

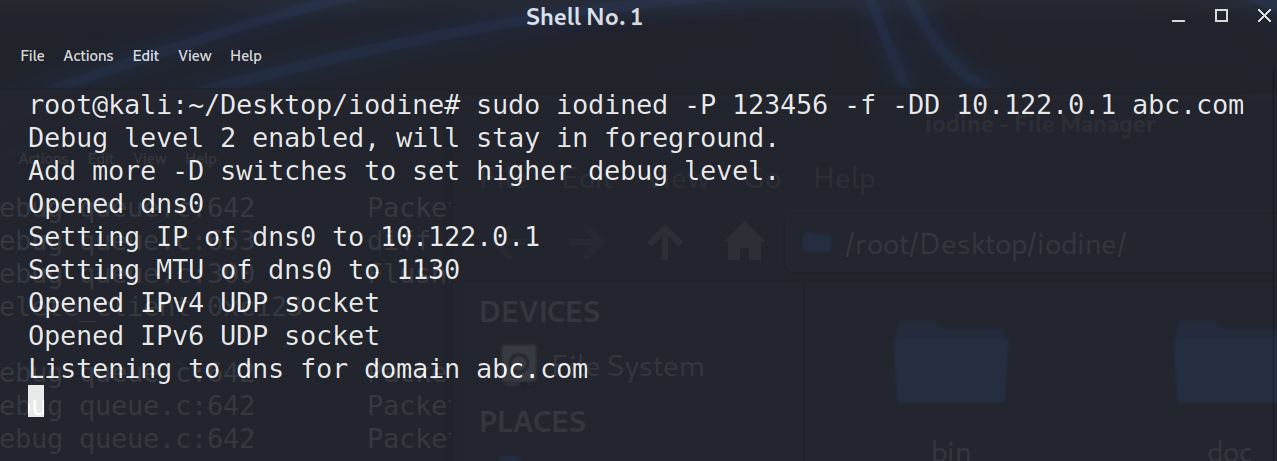
（1）首先在ubuntu20.04中通过git安装iodine，作为被控端。



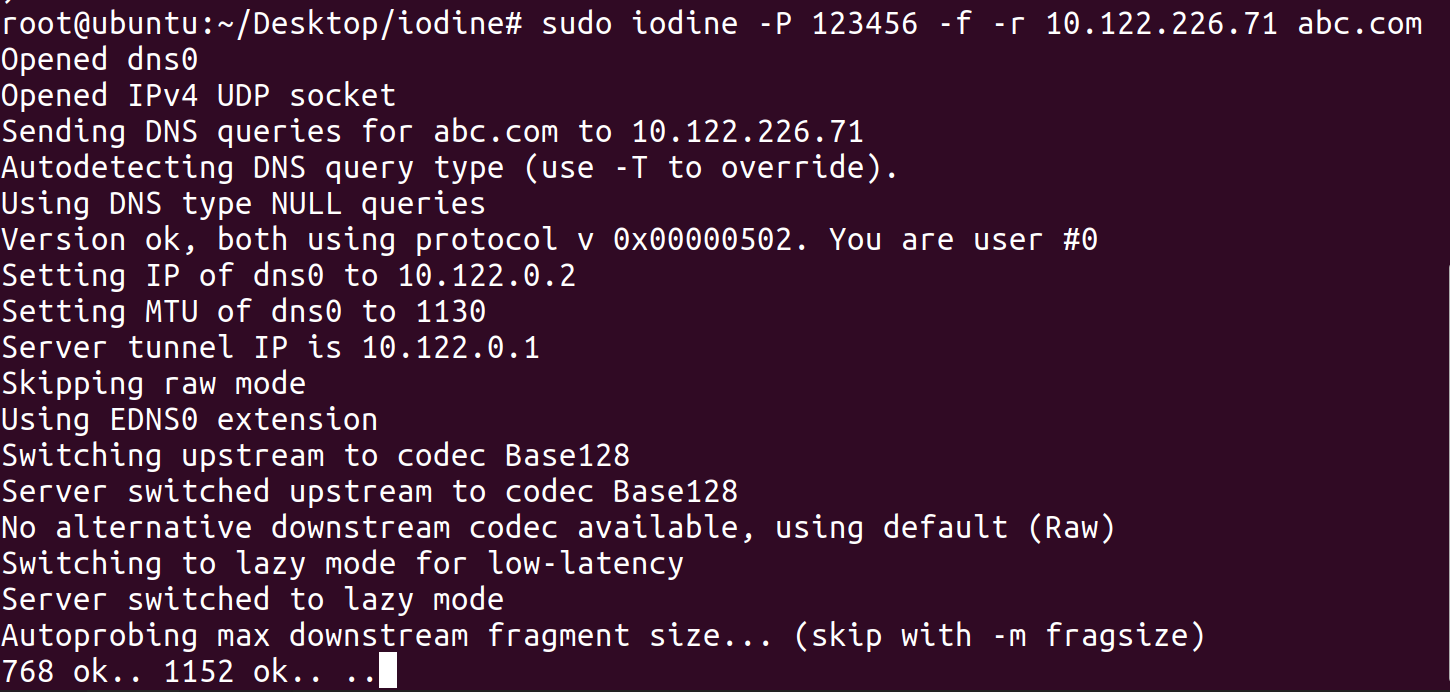
（2）然后在kali中同样安装iodine，作为攻击端。



（3）在攻击端的iodine文件夹下启动命令：**sudo iodined -P 123456 -f -DD 10.122.0.1 abc.com**

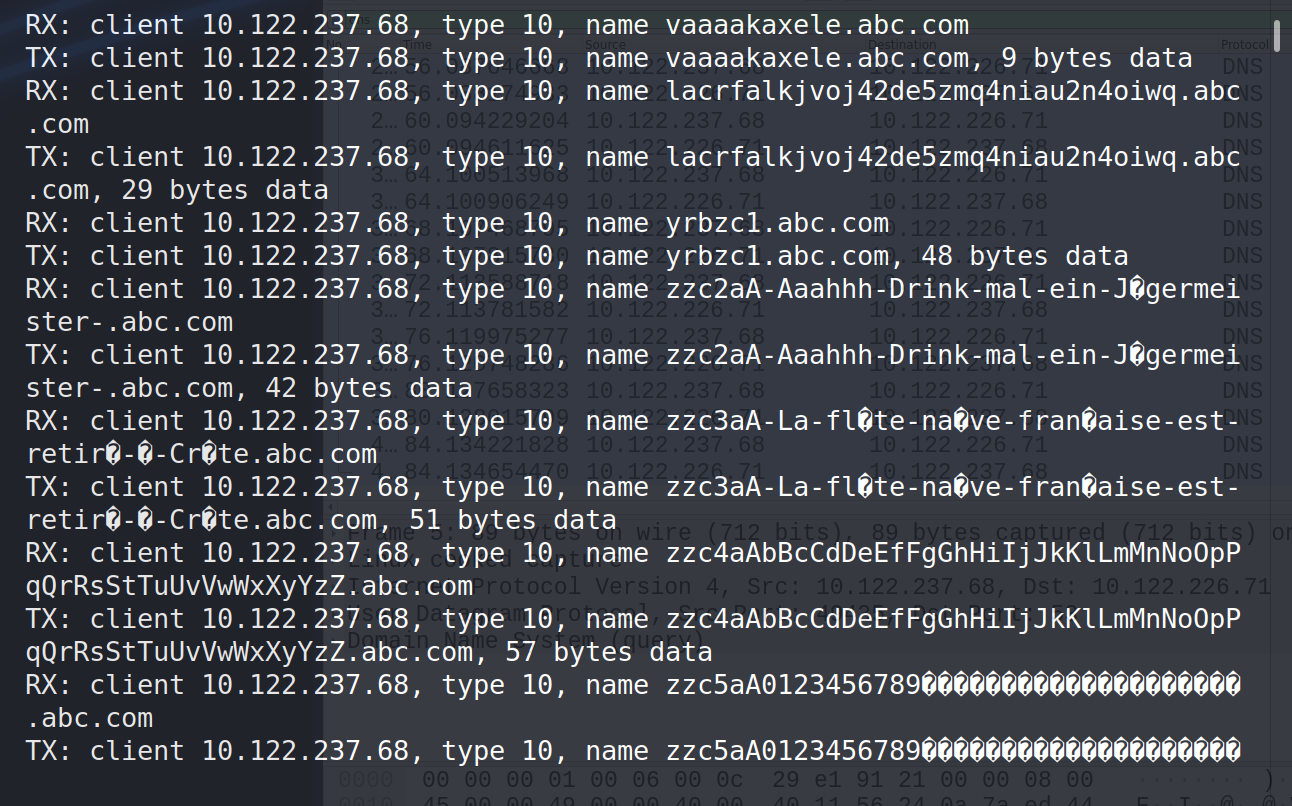


（4）在被控端的iodine文件夹下启动命令：**sudo iodine -P 123456 -f -r 10.122.226.71 abc.com**

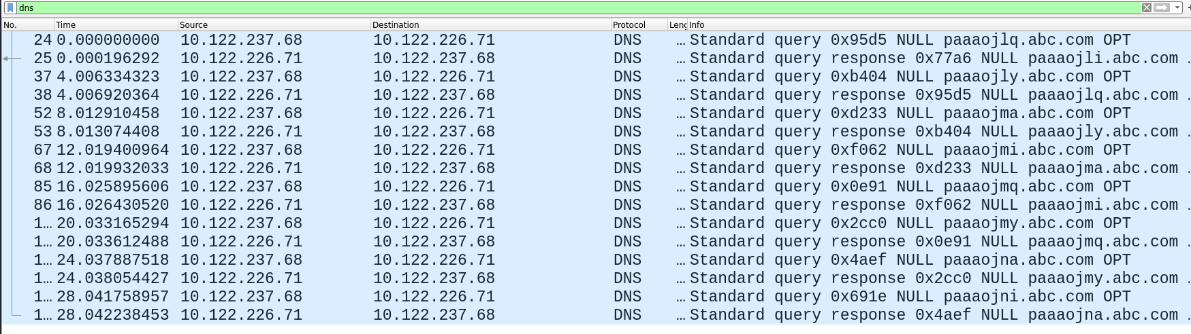


（5）其中**abc.com**为自定义的DNS传输的主域名；**10.122.226.71**为攻击端的ip地址。**123456**为自定义的密码，客户端需要密码才能连接。

（6）连接成功后，攻击端输出如下信息，说明成功建立dns隐蔽信道。



（7）Wireshark抓包结果如下，可以看到以abc.com结尾的域名信息，这些数据就是iodine工具所构造的dns信道数据。



#### 3.1.3 实时检测中的DNS流量获取

###### 3.1.3.1 思路

使用python scapy包中的sniff()函数进行捕获网络数据包，设置过滤规则为端口号为53，且为udp。每20秒保存一次到指定文件夹，并以当前时间命名该pcap数据包文件。并设置其有效期为1小时，这样当实时检测时，就会过滤掉时间早的数据包，避免程序复杂度过大或检测有误。

当用户想终止时，在控制台输入命令即可。所以采用多线程的方式，三个线程同时运行（用户输入命令、实时抓包、实时分析检测）

###### 3.1.3.2 算法

* 抓包保存

1. **def** realtime\_sniff():
2. **global** flag  # 公共变量用于多线程中的”锁“
3. **while** flag:
4. # 设置过滤规则，端口号为53，且为udp数据包  # 每20秒保存一个
5. \_packet = sniff(store=1, filter='port 53 && udp', timeout=20)
6. time\_now = datetime.datetime.now()
7. # 保存为pcap文件 命名为当前时间
8. wrpcap("../RealTimePacket/" + datetime.datetime.strftime(time\_now,
9. '%Y-%m-%d-%H-%M-%S') + '.pcap', \_packet)

* 文件处理

1. **def** get\_file():
2. **global** flag  # 公共变量用于多线程中的”锁“
3. **while** flag:
4. **for** root, dirs, files **in** os.walk("../RealTimePacket/"):
5. **for** f **in** files:
6. file\_time\_str = os.path.join(f)[:19]
7. **try**:
8. # 获取文件时间，判断是否过期
9. file\_time = datetime.datetime.strptime(file\_time\_str, '%Y-%m-%d-%H-%M-%S')
10. **except**:
11. file\_time = datetime.datetime.now()
12. os.remove(os.path.join(root, f))        # 如果数据包命名不合法，删除此数据包
13. now\_time = datetime.datetime.now()
14. **if** (now\_time - file\_time).seconds > 3600:   # 如果数据包时间大于1小时，删除此数据包
15. os.remove(os.path.join(root, f))
16. **print**(file\_time\_str)
17. time.sleep(15)

* 用户命令输入

1. **def** cmd\_exit():
2. **global** flag  # 公共变量用于多线程中的”锁“
3. **while** True:
4. cmd = input()
5. **if** cmd == 'exit':
6. flag = False
7. **break**
8. **else**:
9. **continue**

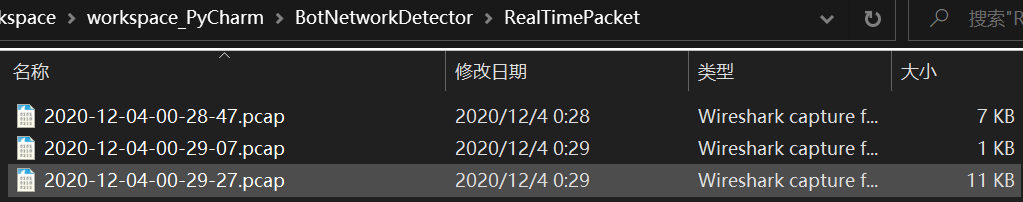
* 多线程

1. flag = True
2. t\_realtime\_sniff = threading.Thread(target=realtime\_sniff)
3. t\_get\_file = threading.Thread(target=get\_file)
4. t\_cmd\_exit = threading.Thread(target=cmd\_exit)
5. t\_realtime\_sniff.start()
6. t\_get\_file.start()
7. t\_cmd\_exit.start()
9. t\_cmd\_exit.join()
10. t\_get\_file.join()
11. t\_realtime\_sniff.join()

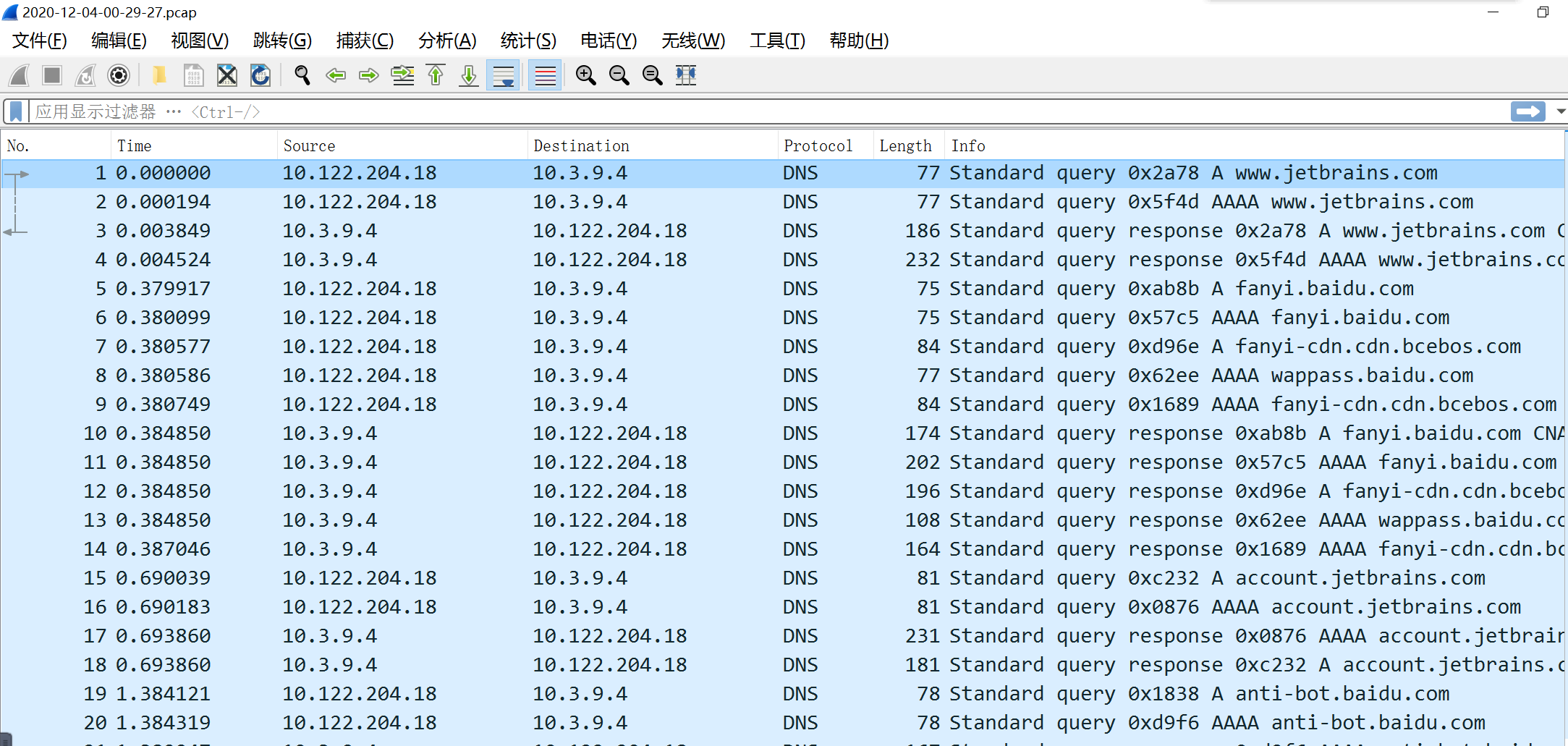
#### 3.1.4 输入输出与性能

这里只列出实时检测中的DNS流量获取的输入输出与性能

用户启动程序，开始抓包、抓到的包文件保存在RealTimePacket文件夹下：



打开其中一个文件，均为dns数据包如下：

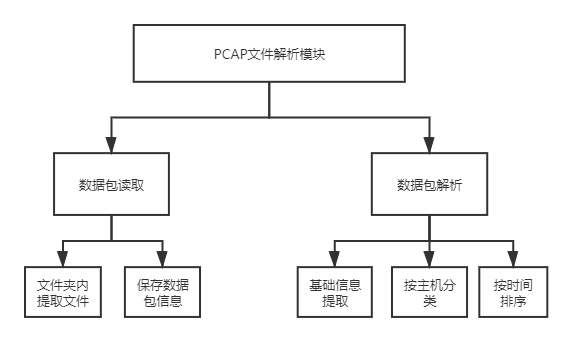


当用户输入exit时，等待线程结束，程序退出。

由于采用多线程处理，实时抓取数据包功能实现较好，采用每20秒（可设置）执行一次线程，性能要求较低，实时性也有所保证。

### 3.2 PCAP文件解析模块

#### 3.2.1 模块功能描述



本模块主要功能为解析pcap文件，根据每一条请求/响应数据包，进行统计，为DNS特征挖掘做准备。

#### 3.2.2 PcapParser类

PcapParser类的功能为pcap文件解析类，保存一些基础信息。下面进行详细说明：

（1）初始化类，以下变量名可以根据命名方式推测含义：

1. **def** \_\_init\_\_(self, \_count, \_mode, folder, \_filename, \_verbose, \_cmd):
2. self.cmd = \_cmd
3. self.count = \_count
4. self.filename = \_filename
5. self.mode = \_mode
6. self.verbose = \_verbose
7. self.dnsUnknownResponseCount = 0
8. self.dnsQueryCount = 0
9. self.dnsResponseCount = 0
10. self.dnsFailedCount = 0
11. self.dnsPktCount = 0
12. self.csv\_obj = CSVWrapper(\_filename, self.cmd)
13. self.progress\_index = 0
14. self.pcapList = []
15. self.fileList = []
16. **for** root, dirs, files **in** os.walk(folder):
17. **for** \_file **in** files:
18. f = open(os.path.join(root, \_file), 'rb')
19. self.fileList.append(f)
20. self.pcapList.append(dpkt.pcap.Reader(f))
21. self.c\_name = ''
22. self.MAX\_PKT\_COUNT = 1000
23. self.start = datetime.datetime.now()

（2）开始解析的整体流程，其中流程都写在了注释里：

1. **def** start\_parse(self):
2. self.csv\_obj.write\_log('1', 'PCAP Processing Started at', str(self.start))
3. self.progress\_index = 10
4. # 将时间戳和包数据分开，一层一层解析，其中ts是时间戳，buf存放对应的包
5. **for** self.pcap **in** self.pcapList:
6. **for** self.ts, self.buf **in** self.pcap:
7. # count表示最大处理数目，这里设置为一千万
8. **if** self.count <= self.dnsPktCount:
9. **break**
10. **else**:
11. # 判断是否为DNS流量
12. **if** 0 == self.check\_dns\_traffic():
13. **continue**
14. # self.udp = self.ip.data
15. # self.ip = self.eth.data
16. **try**:
17. self.dns = dpkt.dns.DNS(self.udp.data)
18. **except**:
19. **continue**
20. # 一共的dns流量数
21. self.dnsPktCount += 1
23. # 是否为DNS查询，统计与request相关的信息
24. **if** self.dns.qr == dpkt.dns.DNS\_Q:
25. self.parse\_request()
26. # 如果不是response，跳过
27. **if** self.dns.qr != dpkt.dns.DNS\_R:
28. **continue**
29. # Opcode:占4位，1/2字节。指示查询种类：0:标准查询；1:反向查询；2:服务器状态查询；3-15:未使用。
30. # 只统计标准查询 DNS\_QUERY
31. **if** self.dns.opcode != dpkt.dns.DNS\_QUERY:
32. **print** (self.dns.opcode)
33. **continue**
34. # RCODE:占4位，1/2字节。由回复时指定的返回码：
35. # 0:无差错；1:格式错；2:DNS出错；3:域名不存在；
36. # 4:DNS不支持这类查询；5:DNS拒绝查询；6-15:保留字段。
37. **if** self.dns.rcode != dpkt.dns.DNS\_RCODE\_NOERR:
38. # 出现错误的响应数，基本没有格式错的
39. self.dnsResponseCount += 1
40. **if** self.dns.rcode == dpkt.dns.DNS\_RCODE\_NXDOMAIN **or** self.dns.rcode == dpkt.dns.DNS\_RCODE\_REFUSED \
41. **or** self.dns.rcode == dpkt.dns.DNS\_RCODE\_SERVFAIL:
42. # 有2,3,5差错的响应数
43. self.dnsFailedCount += 1
44. # 每条数据中的response中的Queries的信息
45. # qd : 向server提出的查询记录
46. **for** answer **in** self.dns.qd:
47. **try**:
48. # 解包，获取目的IP
49. dstIP = struct.unpack('!L', self.ip.dst)[0]
50. **if** self.mode > 0:
51. # 更新response
52. \_ts\_ = str(datetime.datetime.utcfromtimestamp(self.ts).strftime('%d/%m/%y %H:%M:%S'))
53. self.update\_response(self.dns.id, self.inet\_ntoa\_wrap(self.ip.dst),
54. self.answer.name, self.answer.type, self.dns.rcode, 0, 0, \_ts\_)
55. **continue**
56. **except**:
57. **continue**
58. **else**:
59. **print** (
60. self.dnsPktCount, '\t', self.dns.id, ' unhandled dns.rcode:', self.dns.rcode)
61. **continue**
62. # an : 回复记录
63. # 回复记录的个数为0，response的个数也加一，如果不是DNS拒绝查询，继续下一条数据
64. **if** len(self.dns.an) < 1:
65. self.dnsResponseCount += 1
66. **if** self.dns.rcode != dpkt.dns.DNS\_RCODE\_REFUSED:
67. **if** self.mode < 2:
68. **print** (
69. self.dnsPktCount, '\t', self.dns.id, '\tResponse \t Unhandled DNSSEC : opcode',
70. self.dns.opcode, '\t Rcode', self.dns.rcode)
71. **continue**
73. # 对每一个响应记录，进行解析统计
74. # an : 回复记录 server回复的资源记录
75. # ns : 权威记录 权威的资源记录
76. # ar : 格外记录 格外的资源记录
77. **try**:
78. c\_name = ''
79. self.dnsResponseCount += 1
80. **for** self.answer **in** self.dns.an:
81. self.parse\_response()
83. **for** self.answer **in** self.dns.ns:
84. self.parse\_response()
86. **for** self.answer **in** self.dns.ar:
87. self.parse\_response()
89. **except**:
90. **print** (
91. self.dnsPktCount, '\t', 'Unhandled Query type : ')

下面是解析的子函数内容部分：

（3）判断是否为DNS流量：

1. # 判断是否为DNS流量
2. **def** check\_dns\_traffic(self):
3. **try**:    # 解包，物理层
4. self.eth = dpkt.ethernet.Ethernet(self.buf)
5. **except**:
6. **return** 0
7. # 解包，网络层，判断网络层是否存在
8. **if** self.eth.type != 2048:
9. **if** self.eth.type != 34525:
10. **return** 0
11. **try**:
12. self.ip = self.eth.data
13. **except**:
14. **return** 0
16. **if** self.ip.p != 17:
17. **return** 0
18. **try**:
19. self.udp = self.ip.data
20. **except**:
21. **return** 0
22. # 解包，判断传输层协议是否是UDP
23. **try**:
24. **if** self.udp.sport != 53 **and** self.udp.dport != 53:
25. **return** 0
26. **except**:
27. **return** 0
28. **return** 1

（4）对请求包进行解析：

1. **def** parse\_request(self):
2. self.dnsQueryCount += 1
3. **for** self.query **in** self.dns.qd:
4. # 是否是在白名单里，但是此时verbose=1 and mode=3，检测时并不运行
5. **if** checkWhiteList(self.query.name) == 1:
6. **if** self.verbose == 1 **and** self.mode < 2:
7. **print** ('White list Presence : ' + self.query.name)
8. **continue**
9. **elif** self.query.type != 0:
10. **try**:
11. # 模式3，增加一条请求消息
12. **if** self.mode == 3:
13. self.csv\_obj.write\_request(self.dns.id, self.inet\_ntoa\_wrap(self.ip.src), self.query.name, self.query.name.count('.') + 1, self.query.type, len(self.query.name), self.inet\_ntoa\_wrap(self.ip.dst), str(datetime.datetime.utcfromtimestamp(self.ts).strftime('%d/%m/%y %H:%M:%S')))
14. **except**:
15. **continue**
16. **else**:
17. **print** (
18. self.dnsPktCount, '\t', 'Unhandled Request')

（5）对响应包进行解析：

1. **def** parse\_response(self):
2. **if** self.answer.type == dpkt.dns.DNS\_A:
3. **try**:
4. **if** self.eth.type == 2048:
5. dstIP = struct.unpack('!L', self.ip.dst)[0]
6. **else**:
7. dstIP = struct.unpack('!QQ', self.ip.dst)[0]
8. **if** self.mode > 0:
9. **if** self.c\_name == '':
10. self.update\_response(self.dns.id, self.inet\_ntoa\_wrap(self.ip.dst), self.answer.name, self.answer.type, self.dns.rcode, self.answer.ttl, socket.inet\_ntoa(self.answer.rdata), str(datetime.datetime.utcfromtimestamp(self.ts).strftime('%d/%m/%y %H:%M:%S')))
11. **else**:
12. self.update\_response(self.dns.id, self.inet\_ntoa\_wrap(self.ip.dst), self.c\_name, self.answer.type, self.dns.rcode, self.answer.ttl, self.inet\_ntoa\_wrap(self.answer.rdata), str(datetime.datetime.utcfromtimestamp(self.ts).strftime('%d/%m/%y %H:%M:%S')))
13. **except**:
14. **elif** self.answer.type == dpkt.dns.DNS\_CNAME:
15. **try**:
16. c\_name = self.answer.name
17. **except**:
18. **print** (self.dnsPktCount, '\t', 'Failed')
19. **elif** self.answer.type == dpkt.dns.DNS\_MX:
20. **try**:
21. **if** self.mode > 0:
22. **if** self.c\_name == '':
23. self.updateResponse(self.csv\_obj, self.dns.id, self.inet\_ntoa\_wrap(self.ip.dst), self.answer.name, self.answer.type, self.dns.rcode, self.answer.ttl, socket.inet\_ntoa(self.answer.rdata), str(datetime.datetime.utcfromtimestamp(self.ts).strftime('%d/%m/%y %H:%M:%S')))
24. **else**:
25. self.updateResponse(self.csv\_obj, self.dns.id, self.inet\_ntoa\_wrap(self.ip.dst), self.cname, self.answer.type, self.dns.rcode, self.answer.ttl, socket.inet\_ntoa(self.answer.rdata), str(datetime.datetime.utcfromtimestamp(self.ts).strftime('%d/%m/%y %H:%M:%S')))
27. **elif** self.answer.type == dpkt.dns.DNS\_NS:
28. **try**:
29. **if** self.mode > 0:
30. **if** self.c\_name == '':
31. **if** self.answer.rlen == 4:
32. self.update\_response(self.dns.id, self.inet\_ntoa\_wrap(self.ip.dst), self.answer.name, self.answer.type, self.dns.rcode, self.answer.ttl, socket.inet\_ntoa(self.answer.rdata), str(datetime.datetime.utcfromtimestamp(self.ts).strftime('%d/%m/%y %H:%M:%S')))
33. **elif** self.answer.rlen == 4:
34. self.update\_response(self.dns.id, self.inet\_ntoa\_wrap(self.ip.dst), self.c\_name, self.answer.type, self.dns.rcode, self.answer.ttl, socket.inet\_ntoa(self.answer.rdata), str(datetime.datetime.utcfromtimestamp(self.ts).strftime('%d/%m/%y %H:%M:%S')))
36. **elif** self.answer.type == dpkt.dns.DNS\_AAAA:
37. **try**:
38. **if** self.mode > 0:
39. **if** self.c\_name == '':
40. self.update\_response(self.dns.id, self.inet\_ntoa\_wrap(self.ip.dst), self.answer.name, self.answer.type, self.dns.rcode, self.answer.ttl, self.inet\_ntoa\_wrap(self.answer.rdata), str(datetime.datetime.utcfromtimestamp(self.ts).strftime('%d/%m/%y %H:%M:%S')))
41. **else**:
42. self.update\_response(self.dns.id, self.inet\_ntoa\_wrap(self.ip.dst), self.c\_name, self.answer.type, self.dns.rcode, self.answer.ttl, self.inet\_ntoa\_wrap(self.answer.rdata), str(datetime.datetime.utcfromtimestamp(self.ts).strftime('%d/%m/%y %H:%M:%S')))
44. **elif** self.answer.type == 12 **or** self.answer.type == 43 **or** self.answer.type == 46 **or** self.answer.type == 48 **or** self.answer.type == 16 **or** self.answer.type == 33 **or** self.answer.type == 6 **or** self.answer.type == 13:

（5）每解析一条数据，向csv文件中新增一条信息，下面的CSVWrapper类即控制写入csv和读取csv文件，算法和流程如下：

1. **class** CSVWrapper:
2. # 控制csv文件的读写和PcapParser联系
3. **def** \_\_init\_\_(self, filename, \_cmd):
4. self.cmd = \_cmd
5. self.pipeline = 0
6. self.my\_req\_cache\_list = []
7. self.my\_res\_cache\_list = []
8. self.h = Network(filename, \_cmd)
9. self.filename = filename
10. **if** self.pipeline:
11. **return**
12. # 三个文件分别代表：请求csv，响应csv，操作日志csv
13. self.req\_outfile = open('output/' + filename + '\_req.csv', 'w')
14. self.res\_outfile = open('output/' + filename + '\_res.csv', 'w')
15. self.log\_outfile = open('output/' + filename + '\_log.csv', 'w')
17. # 向请求csv中新增一条信息
18. **def** write\_request(self, tid, reqIP, reqURL, nbrDomainToken, querytype, urlLength, dns\_server\_ip, timestamp):
19. **if** len(self.my\_req\_cache\_list) != 3:
20. self.my\_req\_cache\_list.append(str(tid) + ',' + str(reqIP) + ',' + str(reqURL).encode('ascii', 'ignore') + ',' + str(nbrDomainToken) + ',' + str(querytype) + ',' + str(urlLength) + ',' + str(timestamp) + ',' + dns\_server\_ip + '\n')
21. **else**:
22. self.req\_outfile.writelines(self.my\_req\_cache\_list)
23. self.my\_req\_cache\_list[0:3] = []
24. self.my\_req\_cache\_list.append(str(tid) + ',' + str(reqIP) + ',' + str(reqURL) + ',' + str(nbrDomainToken) + ',' + str(querytype) + ',' + str(urlLength) + ',' + str(timestamp) + ',' + dns\_server\_ip + '\n')
26. # 向响应csv中新增一条信息
27. **def** write\_response(self, tid, reqIP, reqURL, reqType, rescode, ttl, resolvedIp, receivedtimestamp):
28. **if** len(self.my\_res\_cache\_list) != 3:
29. self.my\_res\_cache\_list.append(str(tid) + ',' + str(reqIP) + ',' + str(reqURL) + ',' + str(reqType) + ',' + str(rescode) + ',' + str(ttl) + ',' + str(resolvedIp) + ',' + str(receivedtimestamp) + '\n')
30. **else**:
31. self.res\_outfile.writelines(self.my\_res\_cache\_list)
32. self.my\_res\_cache\_list[0:3] = []
33. self.my\_res\_cache\_list.append(str(tid) + ',' + str(reqIP) + ',' + str(reqURL) + ',' + str(reqType) + ',' + str(rescode) + ',' + str(ttl) + ',' + str(resolvedIp) + ',' + str(receivedtimestamp) + '\n')
35. # 新增日志文件
36. **def** write\_log(self, sno, key, val):
37. self.log\_outfile.write(sno + ',' + key + ',' + val + '\n')
39. # 根据用户命令把csv文件关闭，并使用多线程方式执行终端/实时检测报告
40. **def** close(self):
41. self.req\_outfile.close()
42. self.res\_outfile.close()
43. self.log\_outfile.close()
44. t1 = Thread(target=map\_analyse\_data, args=(self.filename, self.cmd))
45. t1.start()
46. t1.join()

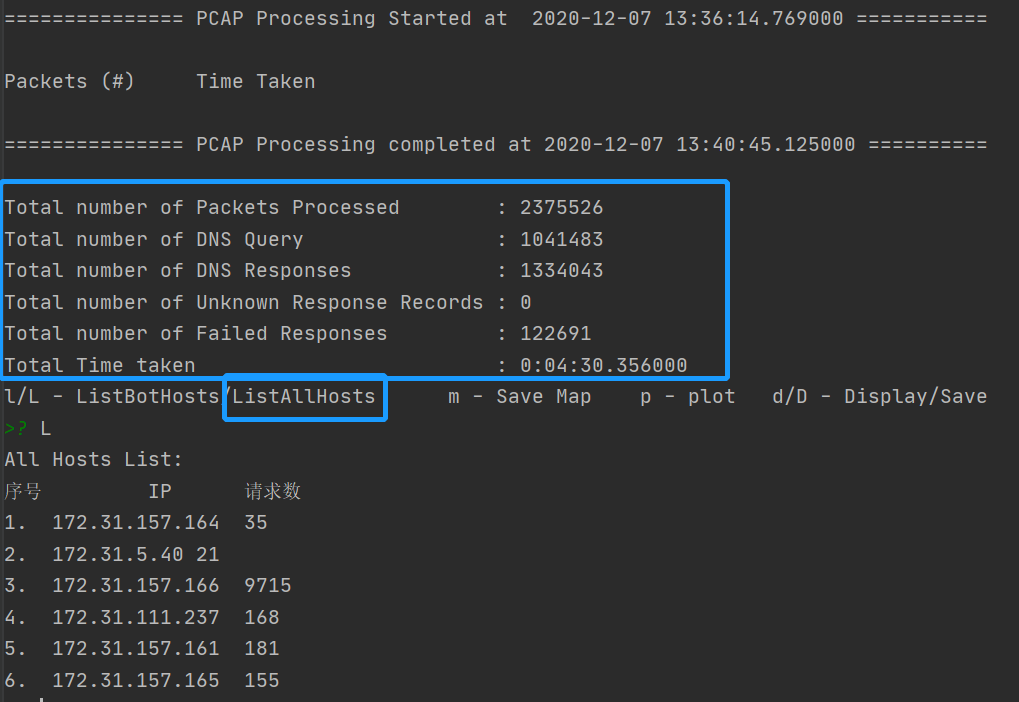
而启动终端/实时检测的部分，可以查看最后两个模块内的内容。

#### 3.2.3 输入

根据用户的命令输入，输入的文件为来自特定文件夹内的所有pcap文件。

#### 3.2.4 输出

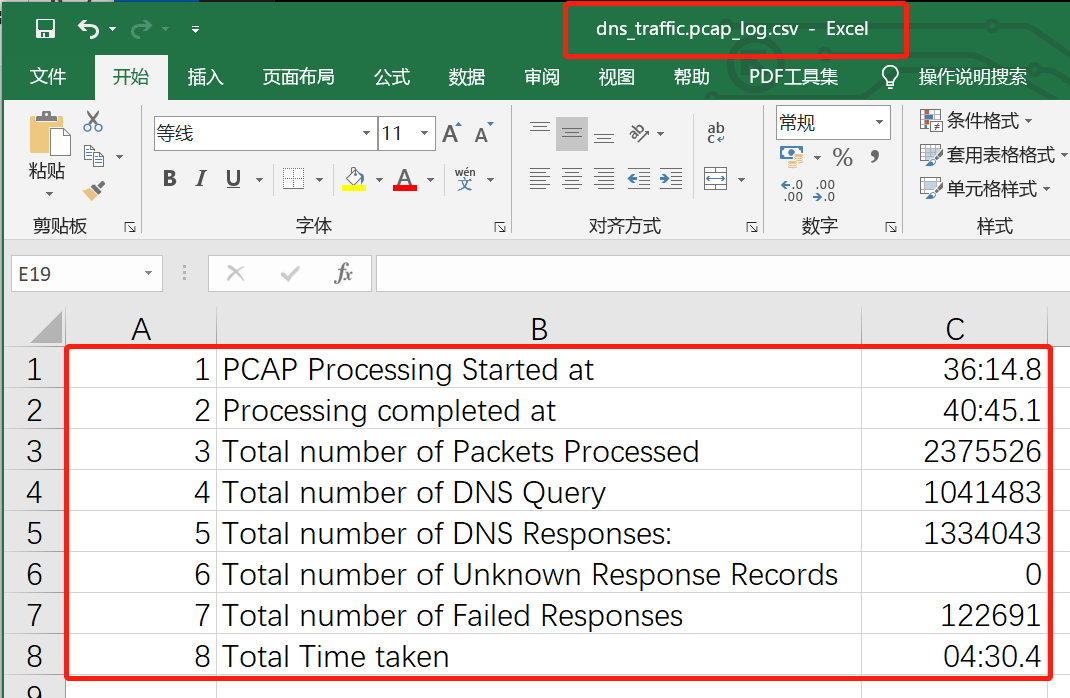
（1）输出pcap文件内的统计信息



（2）输出的csv文件如：

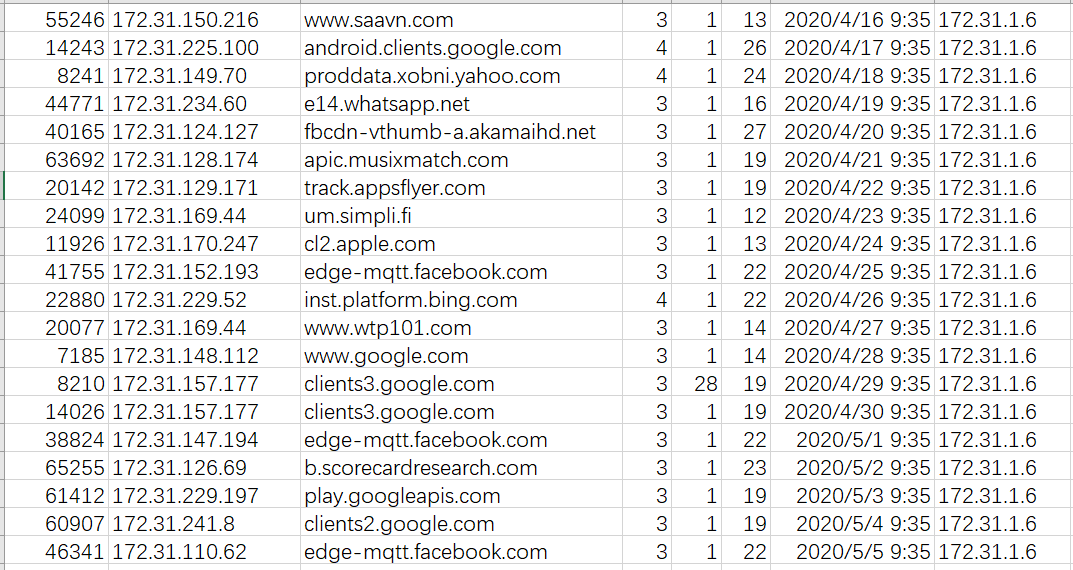
a）log文件

包括上述输出信息



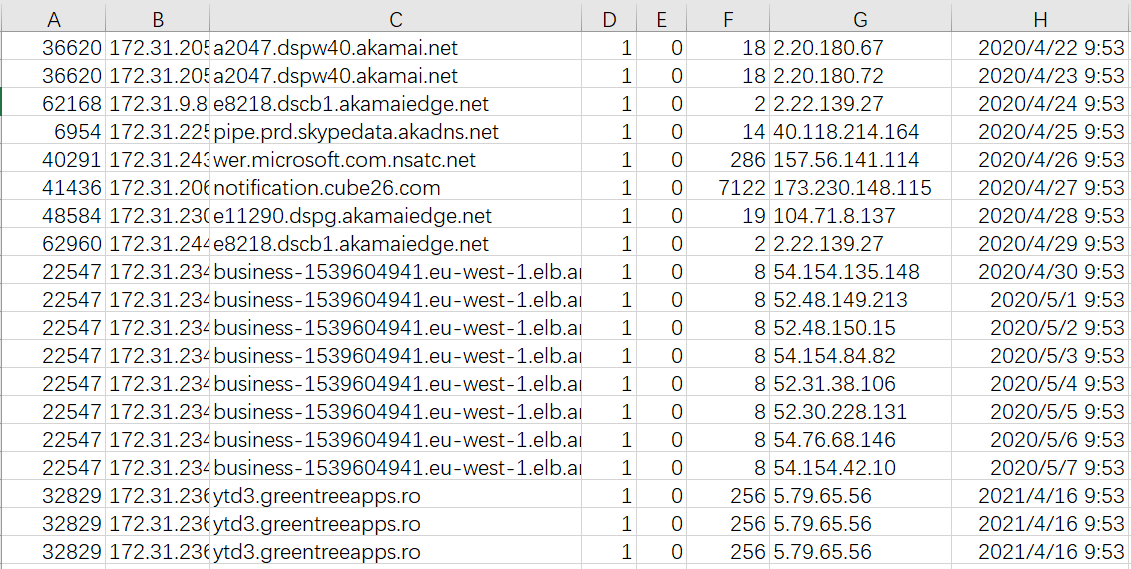
b）request统计文件

包括原目的IP，请求数，错误请求数等



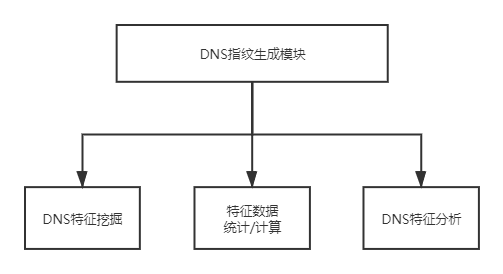
c）response统计文件

同request，输出文件如下为例：



### 3.3 DNS指纹生成模块

#### 3.3.1 模块功能描述



本模块的主要功能为进行DNS特征的提取，保存为CSV文件。

#### 3.3.2 DNS特征提取

经过DNS特征的提取和过滤，我们最终选取的以下15个特征值作为机器学习以及主机分类。



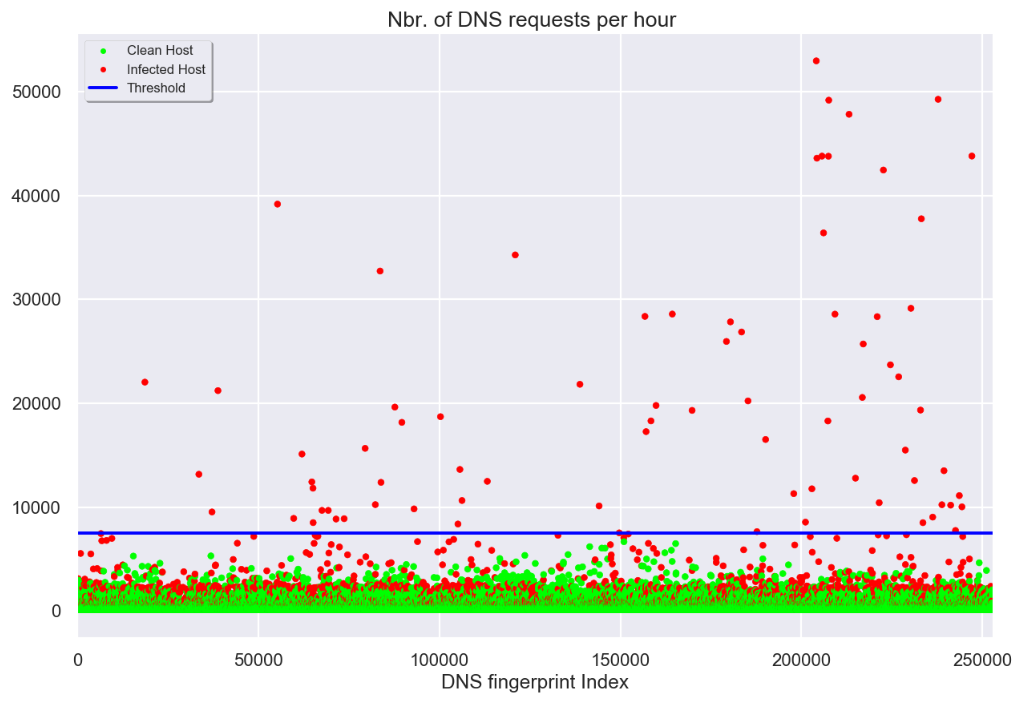
下面的算法是如何提取这15个特征值，其来源为上一个模块的PcapParser文件的输出以及PcapParser类的内容，提取后即写入csv文件：

1. **def** get\_features(self):
2. self.getUFID()
3. host\_cnt = self.UFID
4. tmp\_str = ''
5. tmp\_str2 = 'Hostname, Req\_cnt, Dist\_Req\_Cnt, high\_req\_cnt\_single\_domain, avg\_req\_per\_min, high\_req\_per\_min, ' \
6. 'cnt\_query\_mx, cnt\_query\_ptr, dist\_dns\_server, dist\_tld, dist\_sld, ' \
7. 'uniqueness\_ratio, res\_failed, dist\_city\_cnt, dist\_country\_cnt, flux\_ratio\n'
8. **for** m\_hosts **in** self.hosts:
9. req\_cnt = 0
10. dist\_req\_cnt = 0
11. high\_req\_cnt\_single\_domain = 0
12. first\_req\_time = 0
13. last\_req\_time = 0
14. res\_cnt = 0
15. min\_stats = {}
16. high\_req\_per\_min = 0
17. cnt\_query\_a = 0
18. cnt\_query\_mx = 0
19. cnt\_query\_ns = 0
20. cnt\_query\_ptr = 0
21. list\_tld = []
22. list\_sld = []
23. res\_rec\_cnt = 0
24. res\_success\_cnt = 0
25. res\_failed = 0
26. avg\_ttl\_value = 0
27. high\_ttl\_value = 0
28. list\_dns\_server\_ip = []
29. list\_res\_ip = []
30. list\_country\_name = []
31. list\_sub\_name = []
32. list\_city\_name = []
33. res\_ip\_cnt = 0
34. host\_cnt += 1
35. obj = self.hosts[m\_hosts]
36. **for** m\_urls **in** obj.domain:
37. tld = self.get\_tld(m\_urls)
38. **if** tld **not** **in** list\_tld:
39. list\_tld.append(tld)
40. sld = self.get\_sld(m\_urls)
41. **if** sld **not** **in** list\_sld:
42. list\_sld.append(sld)
43. dist\_req\_cnt += 1
44. token\_list = obj.domain[m\_urls]
45. **if** high\_req\_cnt\_single\_domain < len(token\_list.list):
46. high\_req\_cnt\_single\_domain = len(token\_list.list)
47. **for** token **in** token\_list.list:
48. req\_cnt += 1
49. request = token\_list.list[token]
50. **if** len(request.response) != 0:
51. res\_cnt += 1
52. seconds = self.get\_sec(request.req\_timestamp[9:17])
53. **if** first\_req\_time == 0:
54. first\_req\_time = seconds
55. **if** seconds < first\_req\_time:
56. first\_req\_time = seconds
57. **else**:
58. **if** seconds > last\_req\_time:
59. last\_req\_time = seconds
60. **if** int(request.req\_type) == 1:
61. cnt\_query\_a += 1
62. **elif** int(request.req\_type) == 2:
63. cnt\_query\_ns += 1
64. **elif** int(request.req\_type) == 15:
65. cnt\_query\_mx += 1
66. **elif** int(request.req\_type) == 12:
67. cnt\_query\_ptr += 1
68. **if** request.dns\_server\_ip **not** **in** list\_dns\_server\_ip:
69. list\_dns\_server\_ip.append(request.dns\_server\_ip)
70. mints = request.req\_timestamp[9:14]
71. **if** mints **in** min\_stats:
72. min\_stats[mints] += 1
73. **else**:
74. min\_stats[mints] = 1
75. **for** res **in** request.response:
76. **if** int(res.res\_code) == 0:
77. res\_success\_cnt += 1
78. **if** int(res.res\_code) == 3:
79. res\_failed += 1
80. res\_rec\_cnt += 1
81. **if** res.resolved\_ip != 0:
82. **if** high\_ttl\_value < int(res.ttl):
83. high\_ttl\_value = int(res.ttl)
84. avg\_ttl\_value += int(res.ttl)
85. **if** res.resolved\_ip **not** **in** list\_res\_ip:
86. list\_res\_ip.append(res.resolved\_ip)
87. obj\_ip = IPDetails(res.resolved\_ip)
88. **if** obj\_ip.country\_name **not** **in** list\_country\_name:
89. list\_country\_name.append(obj\_ip.country\_name)
90. **if** obj\_ip.sub\_name **not** **in** list\_sub\_name:
91. list\_sub\_name.append(obj\_ip.sub\_name)
92. **if** obj\_ip.city\_name **not** **in** list\_city\_name:
93. list\_city\_name.append(obj\_ip.city\_name)
95. **for** items **in** min\_stats:
96. **if** min\_stats[items] > high\_req\_per\_min:
97. high\_req\_per\_min = min\_stats[items]
99. **if** res\_success\_cnt != 0:
100. avg\_ttl\_value /= res\_success\_cnt
101. **if** last\_req\_time - first\_req\_time > 60:
102. avg\_req\_per\_min = req\_cnt / ((last\_req\_time - first\_req\_time) / 60)
103. **else**:
104. avg\_req\_per\_min = req\_cnt
105. dist\_city\_cnt = len(list\_city\_name)
106. dist\_country\_cnt = len(list\_country\_name)
107. dist\_dns\_server = len(list\_dns\_server\_ip)
108. dist\_sld = len(list\_sld)
109. dist\_tld = len(list\_tld)
110. tmp\_list = self.filename.split('/')
111. uuid = tmp\_list[(len(tmp\_list) - 1)]
113. flux\_ratio = 0
114. uniqueness\_ratio = req\_cnt / dist\_req\_cnt
115. tmp\_str += m\_hosts + '\_' + uuid[4:8] + uuid[9:11] + ','
116. tmp\_str += str(req\_cnt) + ','
117. tmp\_str += str(dist\_req\_cnt) + ','
118. tmp\_str += str(high\_req\_cnt\_single\_domain) + ','
119. tmp\_str += str(avg\_req\_per\_min) + ','
120. tmp\_str += str(high\_req\_per\_min) + ','
121. tmp\_str += str(cnt\_query\_mx) + ','
122. tmp\_str += str(cnt\_query\_ptr) + ','
123. tmp\_str += str(dist\_dns\_server) + ','
124. tmp\_str += str(dist\_tld) + ','
125. tmp\_str += str(dist\_sld) + ','
126. tmp\_str += str(uniqueness\_ratio) + ','
127. tmp\_str += str(res\_failed) + ','
128. tmp\_str += str(dist\_city\_cnt) + ','
129. tmp\_str += str(dist\_country\_cnt) + ','
130. tmp\_str += str(flux\_ratio) + '\n'
132. self.updateUFID(host\_cnt)
133. csv\_file\_name = '..\\Output\\DNS\_FP.csv'
134. **if** os.path.exists(csv\_file\_name):
135. os.remove(csv\_file\_name)
136. csv\_outfile = open(csv\_file\_name, 'w')
137. tmp\_str = tmp\_str2 + tmp\_str
138. csv\_outfile.write(tmp\_str)
139. csv\_outfile.close()

#### 3.3.3 DNS特征解释

下面将针对这15个特征值做详细分析：下面每个特征分析中有一幅图，代表每个特征的散点图，输入为已经分类好的流量和每个主机的特征值，横坐标为样本的序号，纵坐标表示对应的特征值。**红色代表Bot僵尸主机，绿色代表良性主机，蓝色的线代表良性主机最大可能达到的值。**

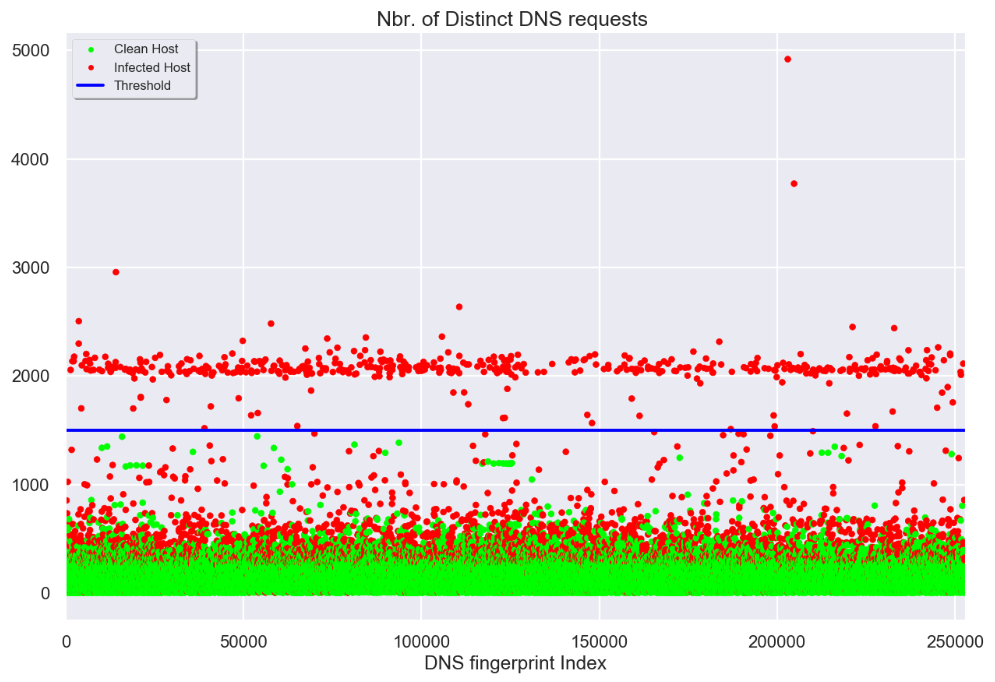
**（1）每小时DNS请求的数量**



受感染的僵尸主机每小时的请求数量往往高于正常主机；从上图可知，绝大多数异常样本的DNS请求数量都高于1200，正常样本的DNS请求数量大多数均低于1200，正常样本最大可能达到的值为7500，是一个很明显的隐蔽信道检测特征。

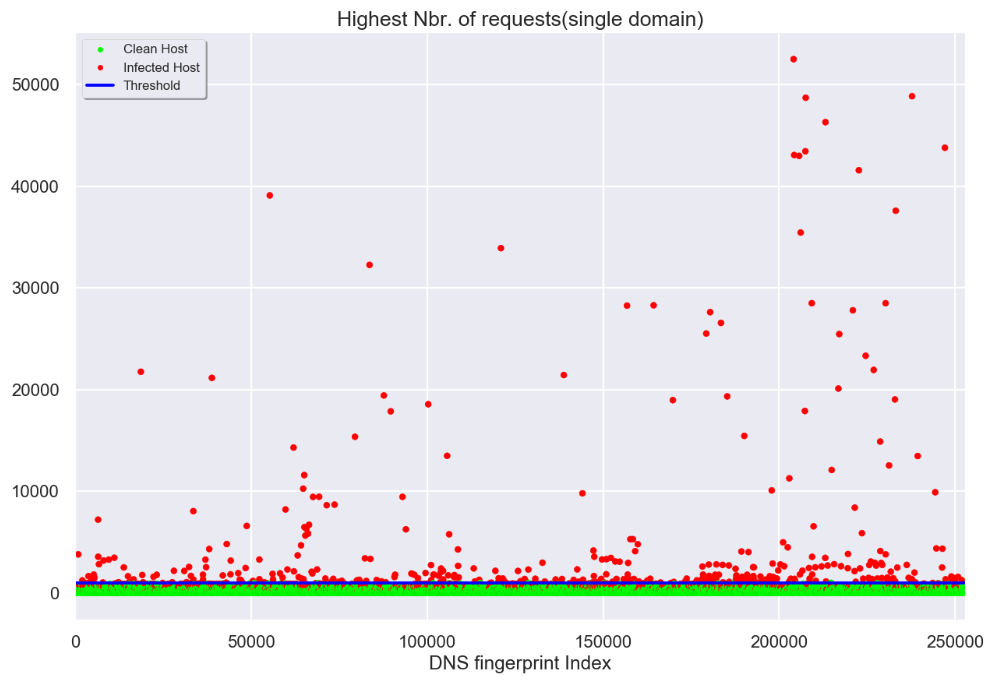
在正常请求过程中，只有在第一次访问域名或本地dns解析结果过期的情况下，才会发起DNS解析请求，因此对于一台主机来说，不会有太过频发的DNS请求；但是对于僵尸主机，需要借助DNS隧道进行通信，所以短时间内会有大量的DNS请求

**（2）每小时不同的DNS请求的数量**



受感染的僵尸主机每小时的请求不同目标数量往往高于正常主机；从上图可知，绝大多数异常样本的请求不同目标数量都高于450，正常样本的请求不同目标数量大多数均低于450。感染DGA恶意软件的主机往往比普通主机具有更多不同的请求以满足其通信需求，因而其不同的DNS请求数便会更多。

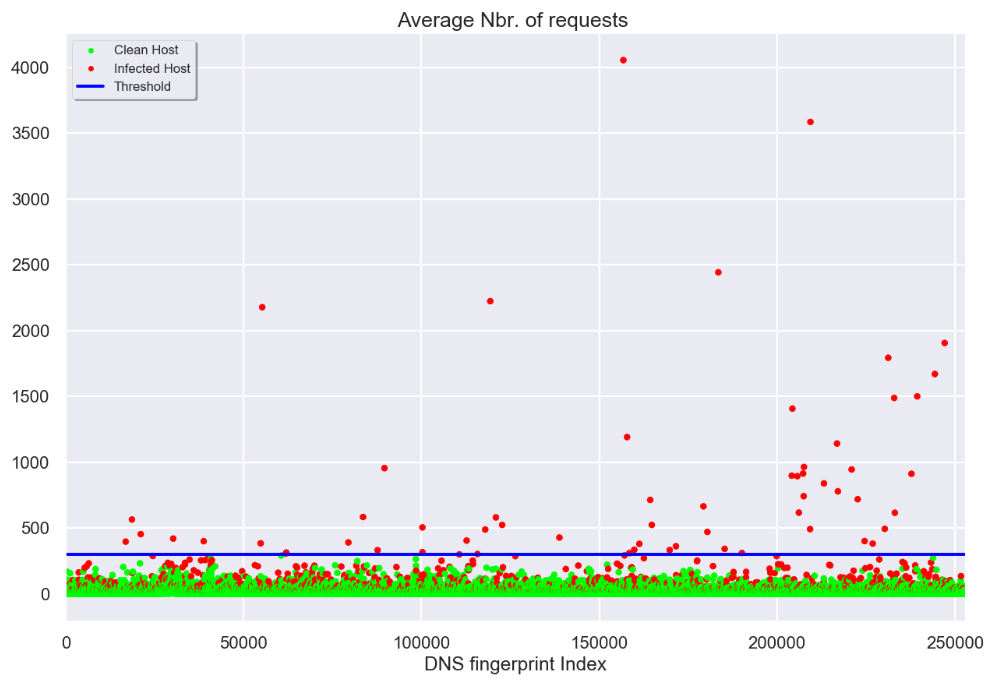
**（3）单个域的最大请求数**



受感染的僵尸主机每小时对于单个域的请求数量往往高于正常主机；从上图可知，绝大多数异常样本对于单个域的请求数量都高于470，正常样本的请求对于单个域的请求数量大多数均低于470。

当受感染主机通过DNS隧道进行通信的时候，会在短时间内对某个DNS权威服务器发送大量的DNS请求信息，以完成指定通信，因此与正常请求相比，对于单个域的最高请求次数是要高的。

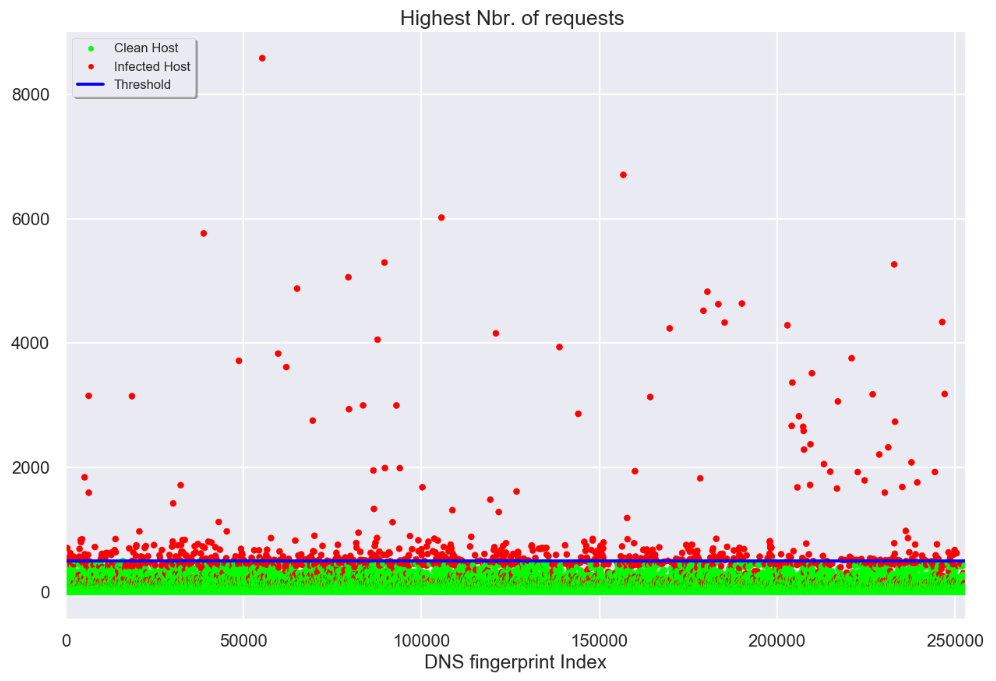
**（5）每分钟平均请求数**



受感染的僵尸主机每小时对于单个域的请求数量往往高于正常主机；从上图可知，绝大多数异常样本对于单个域的请求数量都高于80，正常样本的请求对于单个域的请求数量大多数均低于80，但除少量样本情况外，二者相差并不大。

用于检测受恶意软件感染的计算机，该计算机不使用短暂的DNS请求，而是使用休眠间隔定期对DNS请求做出贡献。这是由该指标的计算方式导致的，它的计算方法是将主机发送的请求数除以主机处于活动状态并使用DNS服务的持续时间。因而尽管恶意主机的请求总数更多，但是它的活跃持续时间也长，所以平均到每分钟，差别就不会太大。

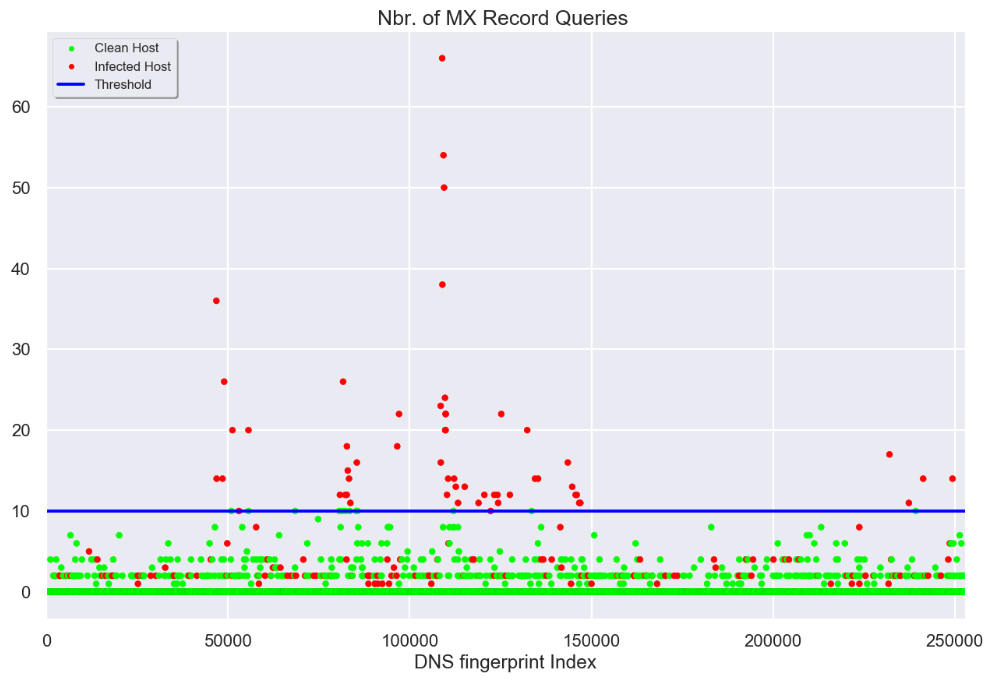
**（5）每分钟最多请求数**



受感染的僵尸主机每分钟最多请求数一般高于正常主机；从上图可以明显的看出，绝大多数正常主机的每分钟最多请求数小于500，而僵尸主机较高，且不均匀。

这个特征值帮助检测感染恶意软件的僵尸程序，这些恶意软件使用短暂的DNS请求通过域生成算法生成的多个URL与C＆C服务器进行通信。

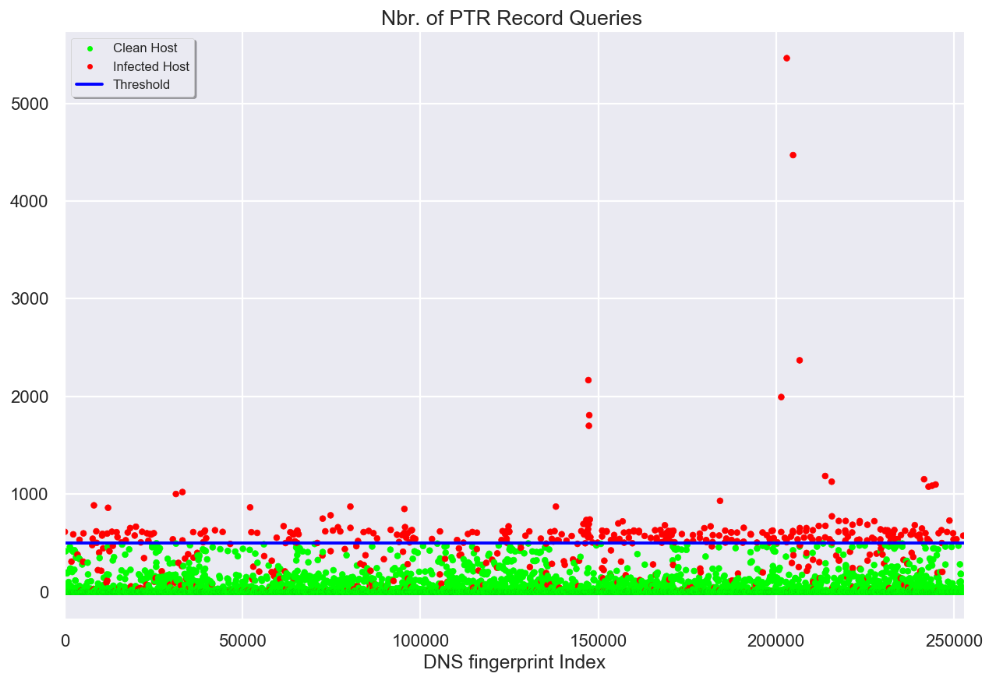
**（6）MX记录查询数**



邮件交换记录 (MX record)是域名系统（DNS）中的一种资源记录类型，用于指定负责处理发往收件人域名的邮件服务器。从上图可以看出，无论是正常样本还是异常样本，其MX记录查询数区别不是很明显。

邮件交换记录查询数是网络中基于垃圾邮件的僵尸网络的强有力指标。在正常请求中，MX记录的查询需求并不大，而受感染主机利用DNS隧道通信时，如果发现有较大的MX记录的查询需求，很有可能是僵尸主机。

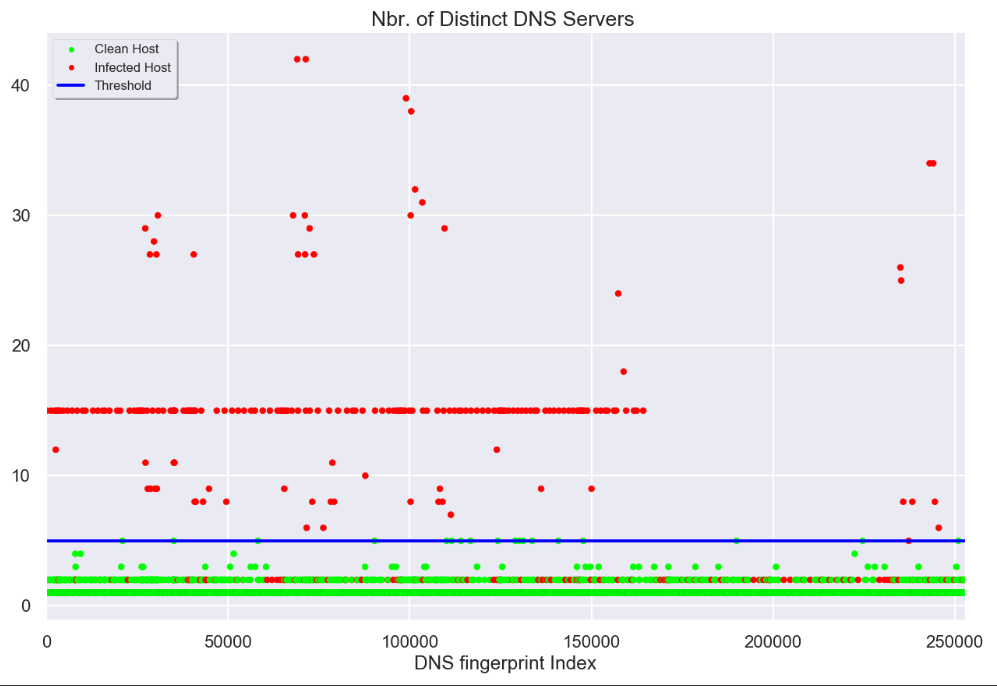
**（7）每小时PTR记录查询数**



PTR记录，是电子邮件系统中的邮件交换记录的一种；另一种邮件交换记录是A记录（在IPv4协议中）或AAAA记录（在IPv6协议中）。PTR记录常被用于反向地址解析。从上图中可以看到大多数情况下，二者PTR查询数量都为0，但是个别情况下异常样本的PTR记录查询数会非常高。

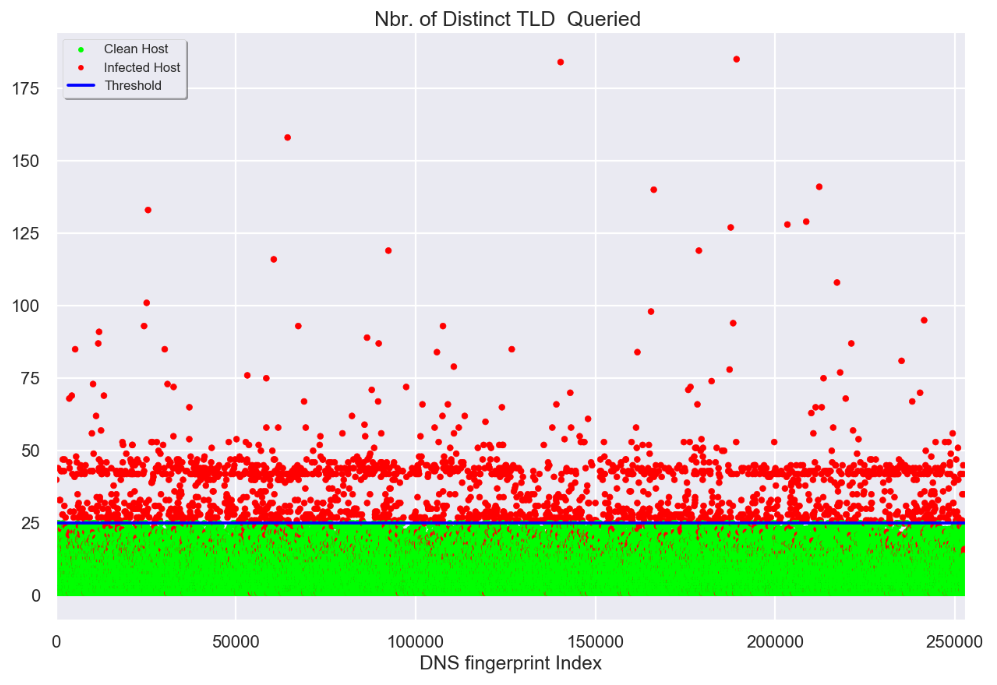
受感染主机在某些情况需要通过ip来反向解析地址，因此会出现这种PTR记录查询数量特别高的情况，但并非一直会有PTR查询的需求。

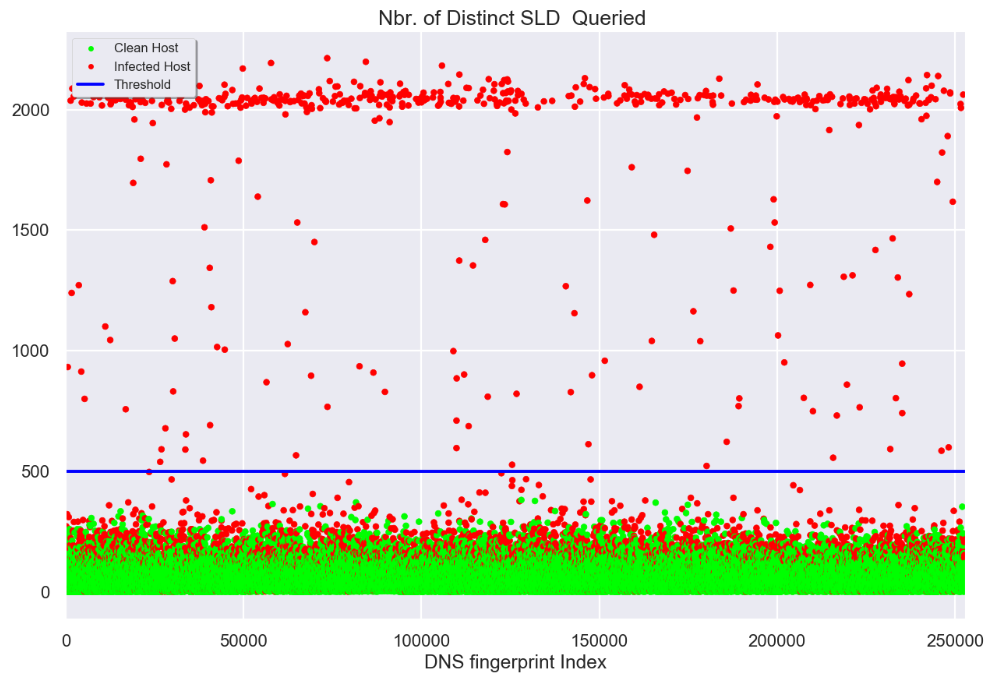
**（8）每小时查询的不同DNS服务器数量**



从上图可知，异常样本对于不同DNS服务器的查询数量要明显高于正常样本。该特征值有助于检测网络中具有异常行为的机器，因为标准系统查询多个DNS服务器的情况并不常见。

**（9）（10）每小时查询的不同TLD/SLD域数量**



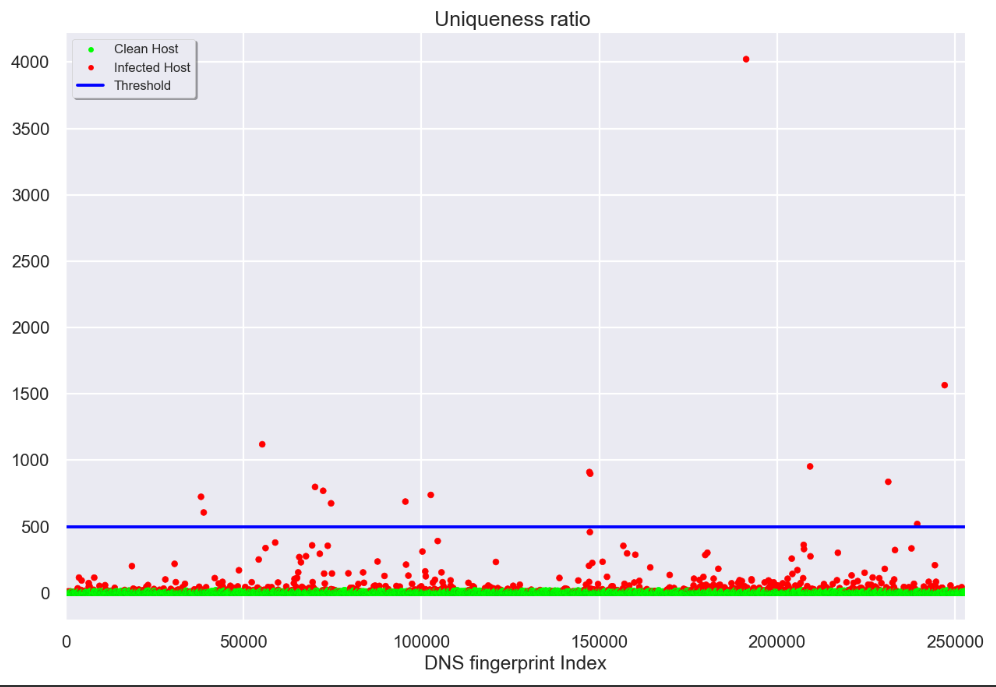


TLD: top level domain，顶级域名

SLD: second level domain，二级域名

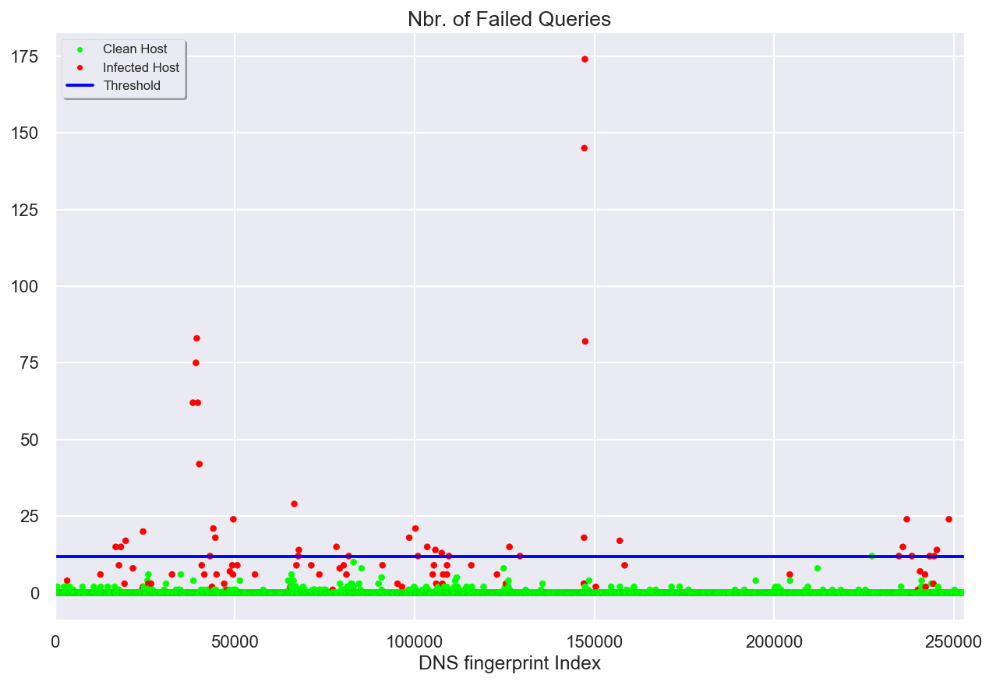
异常样本对于不同TLD/SLD域的查询数量要明显高于正常样本。该特征在在检测基于DGA的机器人方面非常有效，因为这些机器人不仅生成具有不同二级域的随机域，还生成具有不同顶级域名的随机域

**（11）发送的请求数与发送的不同请求数之比**



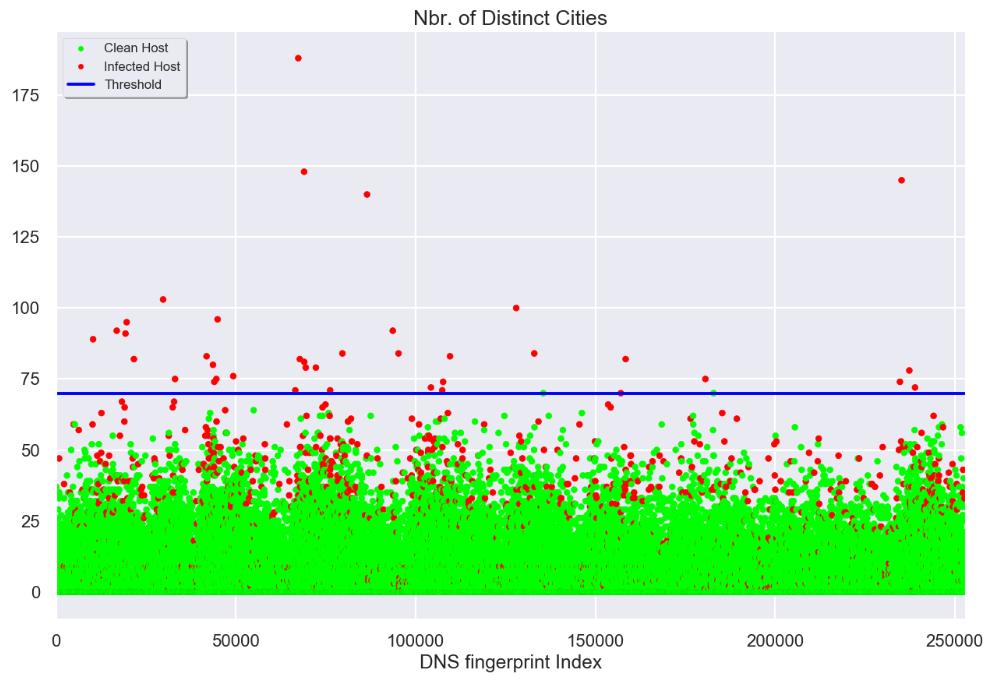
也叫作唯一性比率，是在主机每小时发送至少1000个请求的假设下，发送的请求数与发送的不同请求数之比。根据上图所示，正常样本的唯一性比率很低，只有个别样本较高，异常主机的唯一性比率较高且很分散，因此可以作为检测异常主机的强烈指标。

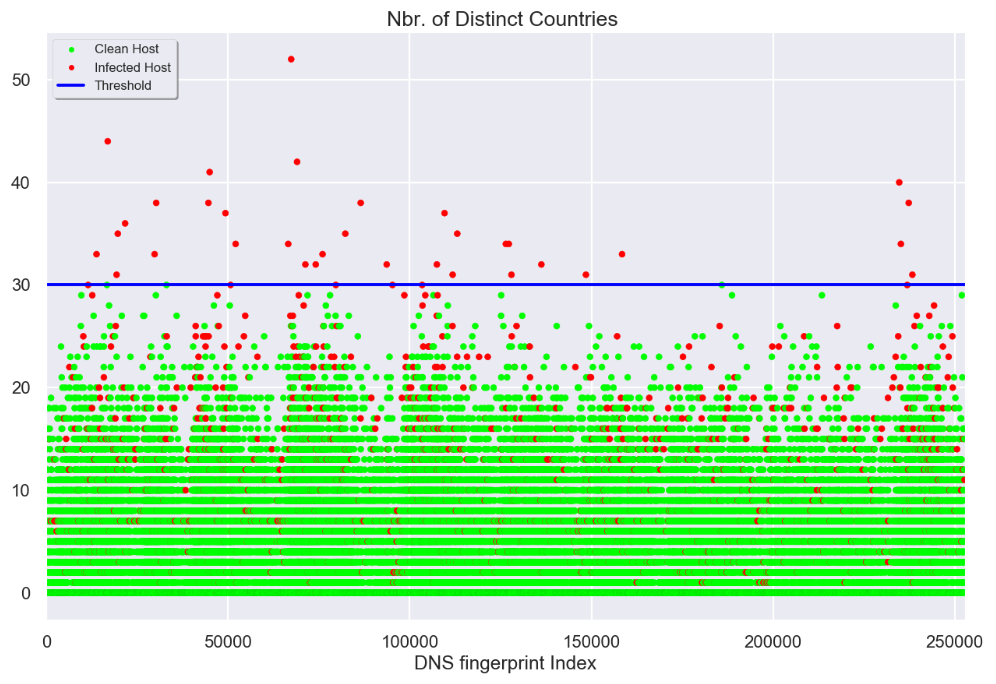
**（12）每小时Failed/NXDOMAIN查询的数量**



每小时Failed/NXDOMAIN查询的数量网络中主机感染的一个非常强的指标。 它通过主机维护响应代码等于DNS\_RCODE\_ NXDOMAIN的响应数。

**（13）（14）解析的IP地址的不同城市/国家的数量**





正常样本大多是解析同一个城市/国家的ip地址，但是异常样本会解析多个城市/国家的ip地址。这是一个强烈的异常指标，尤其是当IP地址分布在各个城市时，受感染主机会针对目标ip进行发送特定请求，因此其解析的ip地址的不同城市的数量就会更多。

**（15）发送的不同请求与解析的不同ip地址的比率**

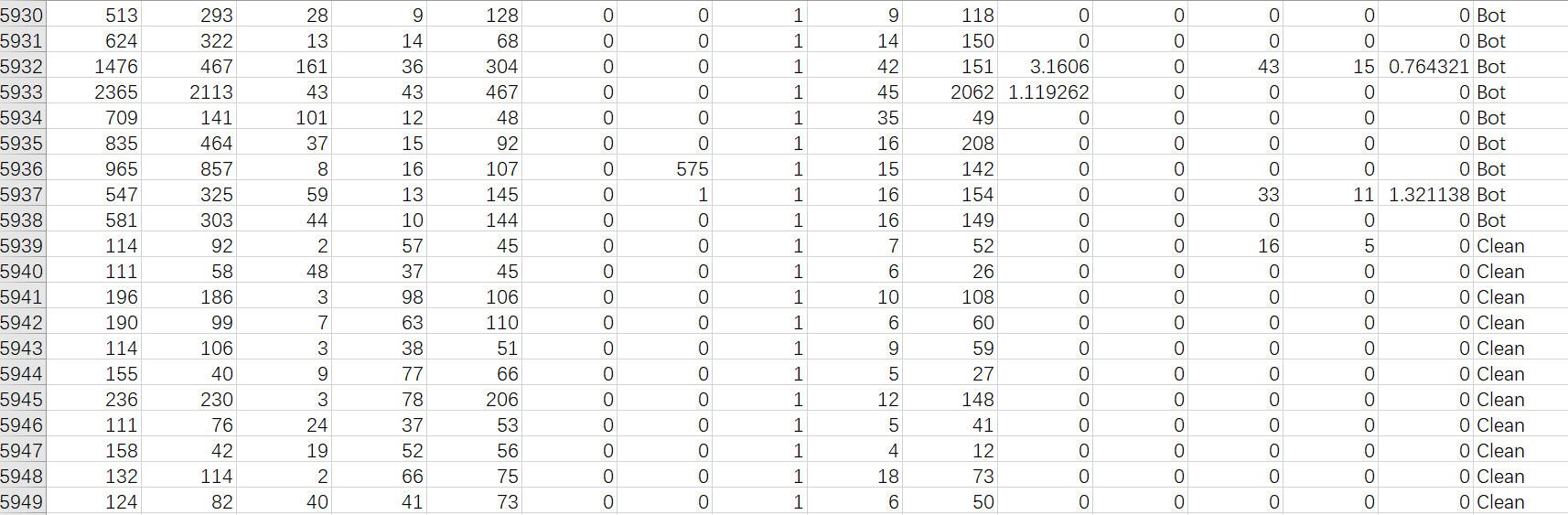


也叫作Flux ratio，在主机发送至少100个查询并且已收到至少100个响应的条件下，发送的不同请求与解析的不同ip地址的比率。

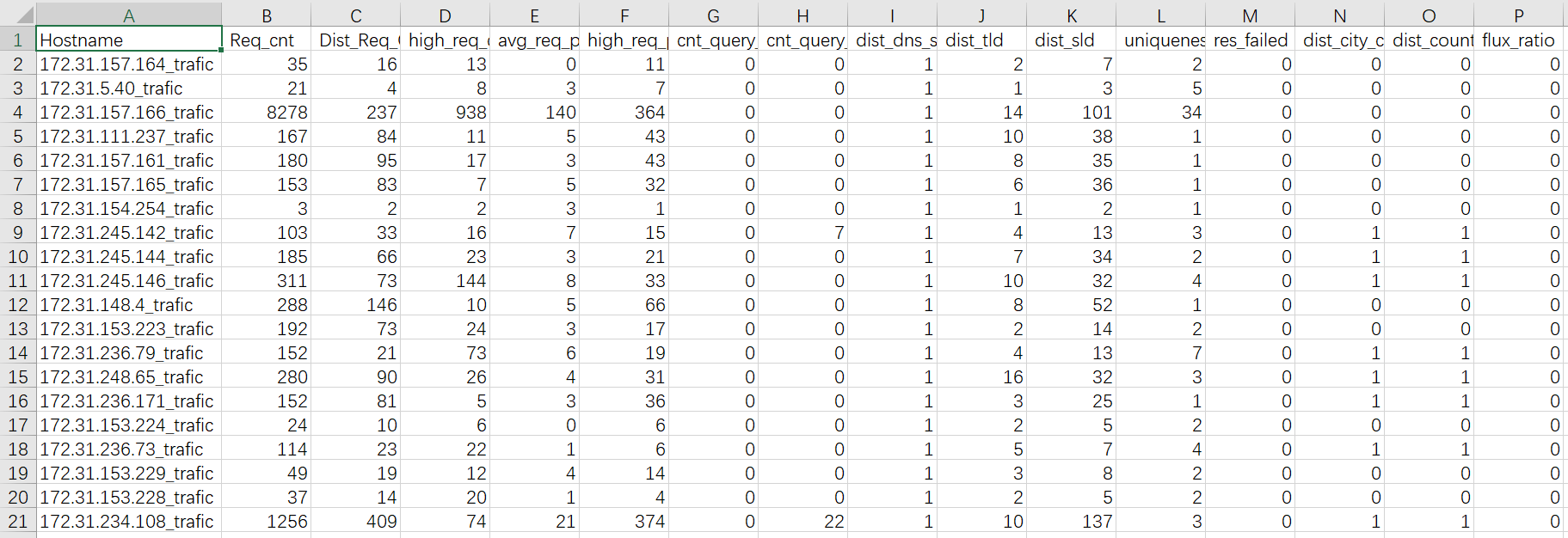
但是根据上图所示，部分正常样本也有较高的Flux ratio，因此以此特征值检测DNS隧道，重要度不是那么高，误报率可能会很高。

#### 3.3.4 模块输出

（1）对于机器学习的样本，输出csv文件，为15个特征值的值和，是否为僵尸主机的标记，如下图所示：

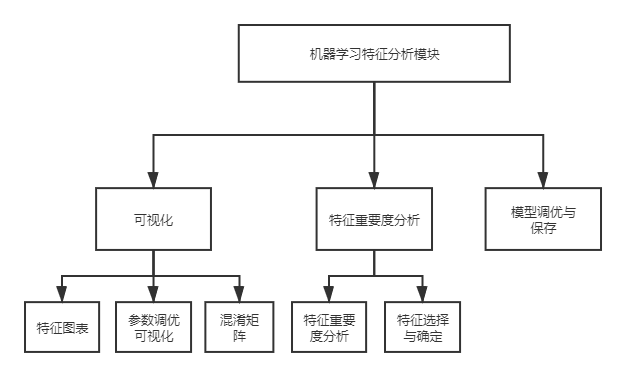


（2）对于待检测的pcap文件，输出csv文件，为主机名+15个特征值的值，用于机器学习分类器的输入。



### 3.4 机器学习特征分析模块

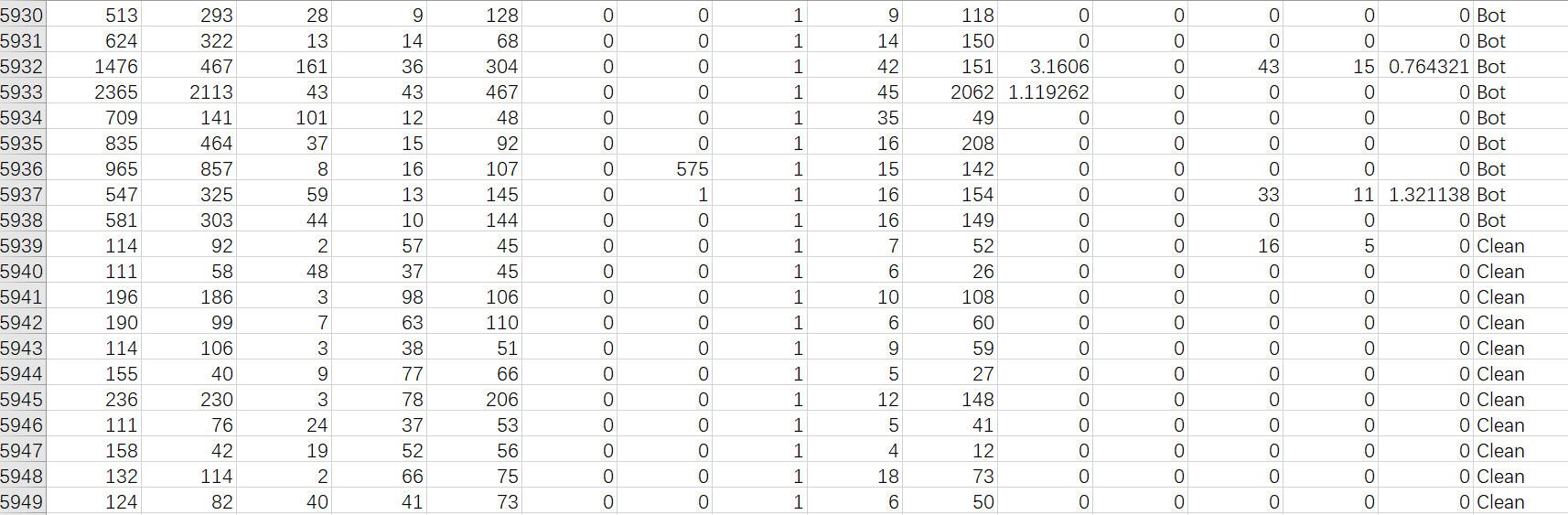
#### 3.4.1 模块功能描述



本模块的功能主要为通过机器学习，生成一个准确度最高的模型，用于分类网络中的DNS流量，包括对特征重要度的分析、特征的选择、模型参数的调整等

#### 3.4.2 输入

机器学习的输入来自上一个模块“DNS指纹生成模块”的输出文件，为csv文件，文件内保存的是15个特征值的数据和已经分类好的正常主机的异常主机的标记，如下图所示：



其中第一列到第15列为选定的15个特征值的数值，最后一列表示分类好的主机标记，Bot为感染了僵尸网络的主机、Clean代表正常主机。

下图为输入文件的统计信息：



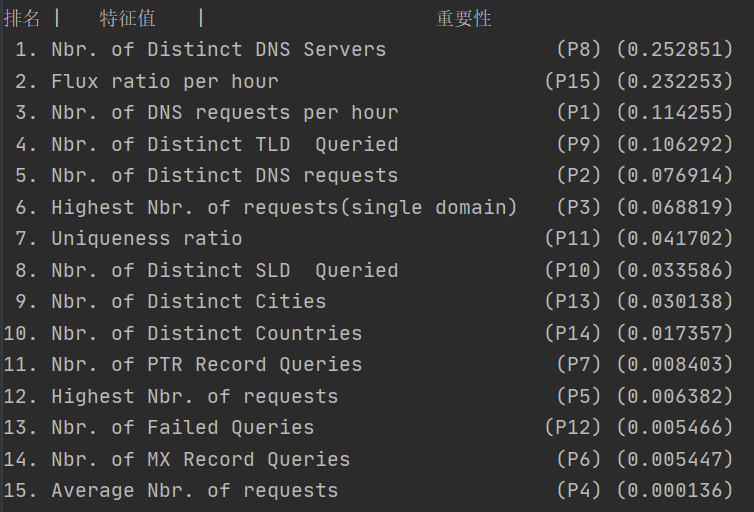
#### 3.4.3 算法

整个过程的算法如下所示，所有的函数功能都详细写在了代码注释里：

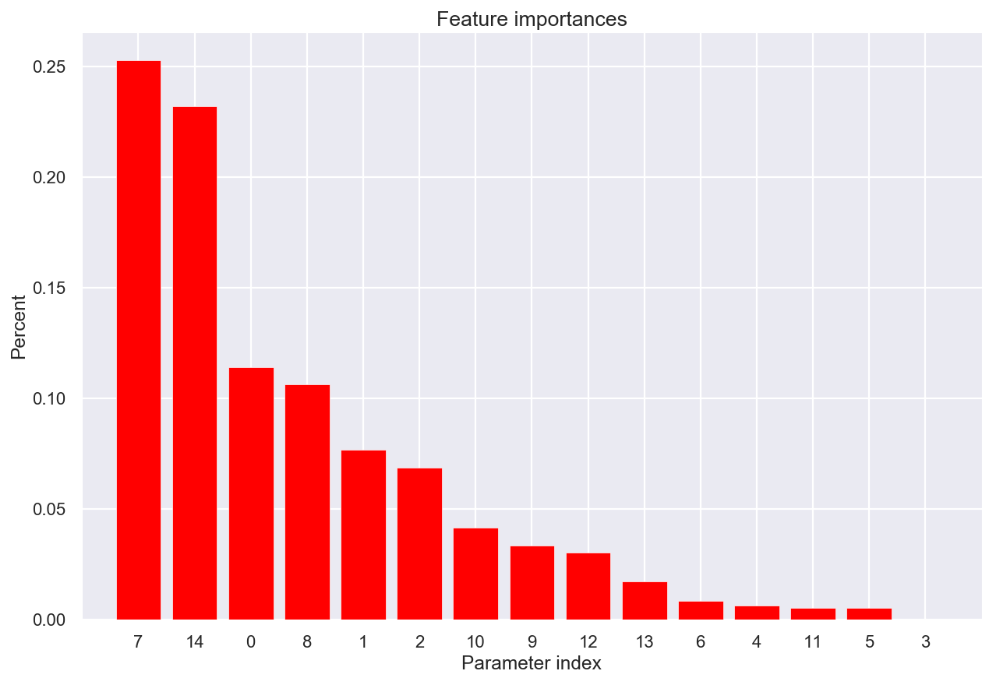
1. **class** BotNetworkDetector:
3. **def** \_\_init\_\_(self):
4. self.df = ""
6. # 读取csv文件，保存到self.df里
7. **def** import\_db(self):
8. # 设置主题风格
9. sns.set\_style("darkgrid")
10. plt.style.use("seaborn-poster")
11. # 加载数据集
12. self.df = pd.read\_csv("export.csv", low\_memory=False)
13. **print**("数据集中一共有%s行，%s 列" % self.df.shape)
15. # y：结果，clean or Bot -> 1 or 0
16. **def** encode\_values(self):
17. self.y = self.df.Result.str.strip()
18. self.labelencoder\_y = LabelEncoder()
19. self.y = self.labelencoder\_y.fit\_transform(self.y)
21. # 统计clean Bot分别有多少
22. **def** display\_count(self):
23. # optional
24. clean = 0
25. Bot = 0
26. **for** row\_index, row **in** self.df.iterrows():
27. **if** ((row.Result.strip()) == 'Clean'):
28. clean = clean + 1
29. **else**:
30. Bot = Bot + 1
31. **print**(clean, Bot)
33. # 数据分类，0.3的测试集
34. **def** split\_data(self):
35. self.df.Result.unique()
36. # P1-P15，得到除了Result以外的所有变量
37. self.x = self.df[self.df.columns.difference(['Result'])].values
38. **print**("自变量X的数目和维度:  %s, %s" % self.x.shape)
39. **print**("因变量Y的数目和维度:  %s, 1" % self.y.shape)
40. **from** sklearn.model\_selection **import** train\_test\_split
41. self.x\_train, self.x\_test, self.y\_train, self.y\_test = train\_test\_split(self.x, self.y, test\_size=0.3, random\_state=42)
43. # 随机森林算法
44. **def** create\_forest(self):
45. # 分类器
46. self.rf\_classifier = RandomForestClassifier(n\_estimators=10, max\_depth=None, min\_samples\_split=2, random\_state=0)
47. # 填充数据
48. self.rf\_classifier.fit(self.x\_train, self.y\_train)
49. # 准确率
50. **print**("[随机森林算法]训练集分数：%s" %self.rf\_classifier.score(self.x\_train, self.y\_train))
51. **print**("[随机森林算法]测试集分数：%s" %self.rf\_classifier.score(self.x\_test, self.y\_test))
53. # 计算每个特征值的importance，权重
54. **def** show\_feature\_imp(self):
55. self.param\_array = [
56. ['P1', 'Nbr. of DNS requests per hour ', 7500],
57. ['P2', 'Nbr. of Distinct DNS requests ', 1500],
58. ['P3', 'Highest Nbr. of requests(single domain)', 1000],
59. ['P4', 'Average Nbr. of requests ', 300],
60. ['P5', 'Highest Nbr. of requests ', 500],
61. ['P6', 'Nbr. of MX Record Queries ', 10],
62. ['P7', 'Nbr. of PTR Record Queries ', 500],
63. ['P8', 'Nbr. of Distinct DNS Servers ', 5],
64. ['P9', 'Nbr. of Distinct TLD  Queried ', 25],
65. ['P10', 'Nbr. of Distinct SLD  Queried ', 500],
66. ['P11', 'Uniqueness ratio ', 500],
67. ['P12', 'Nbr. of Failed Queries', 12],
68. ['P13', 'Nbr. of Distinct Cities', 70],
69. ['P14', 'Nbr. of Distinct Countries ', 30],
70. ['P15', 'Flux ratio per hour ', 100]]
72. self.importances = self.rf\_classifier.feature\_importances\_
73. std = np.std([self.rf\_classifier.feature\_importances\_ **for** tree **in** self.rf\_classifier.estimators\_],
74. axis=0)
75. indices = np.argsort(self.importances)[::-1]
76. # 打印出特征值的权重
77. **print**("排名\t|\t特征值\t|\t\t\t\t\t重要性")
78. **for** f **in** range(self.x.shape[1]):
79. **print**("%2d. %-40s %5s (%f)" % (
80. f + 1, self.param\_array[indices[f]][1], "(P" + str(indices[f] + 1) + ")", self.importances[indices[f]]))
81. # 将特征值排序并以柱状图画出来
82. plt.figure()
83. plt.title("Feature importances")
84. plt.bar(range(self.x.shape[1]), self.importances[indices],
85. color="r", yerr=std[indices], align="center")
86. plt.xticks(range(self.x.shape[1]), indices)
87. plt.xlim([-1, self.x.shape[1]])
88. plt.ylabel('Percent')
89. plt.xlabel('Parameter index')
91. plt.show()
92. """
93. 此函数打印并绘制混淆矩阵。
94. 可以通过设置“ normalize = True”来应用规范化。
95. """
96. **def** plot\_confusion\_matrix(self, cm, classes,
97. normalize=False,
98. title='Confusion matrix',
99. cmap=plt.cm.Blues):
100. plt.imshow(cm, interpolation='nearest', cmap=cmap)
101. plt.title(title)
102. tick\_marks = np.arange(len(classes))
103. plt.xticks(tick\_marks, classes, rotation=0)
104. plt.yticks(tick\_marks, classes)
105. **if** normalize:
106. cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
107. **print**("规范化后的混淆矩阵")
108. **else**:
109. **print**('未规范化的混淆矩阵')
110. **print**(cm)
111. thresh = cm.max() / 2.
112. **for** i, j **in** itertools.product(range(cm.shape[0]), range(cm.shape[1])):
113. plt.text(j, i, cm[i, j], fontsize=20,
114. horizontalalignment="center",
115. color="white" **if** cm[i, j] > thresh **else** "black")
117. plt.tight\_layout()
118. plt.ylabel('Actual')
119. plt.xlabel('Predicted')
120. plt.show()
122. # （二分类模型）绘制预测模型，准确率
123. # https://baijiahao.baidu.com/s?id=1619821729031070174&wfr=spider&for=pc 模型评估之混淆矩阵（confusion\_matrix）含义
124. **def** plot\_predicted\_model(self):
125. # Predicted Model
126. self.y\_pred = self.rf\_classifier.predict(self.x\_test)
127. self.cm = confusion\_matrix(y\_true=self.y\_test, y\_pred=self.y\_pred)
128. # self.plot\_confusion\_matrix(self.cm, self.labelencoder\_y.classes\_)
129. # TN = 73785    # (True Negative)   ：将负类预测为负类数,真实为1,预测为1
130. # FN = 294      # (False Negative)  ：将正类预测为负类数,真实为0,预测为1
131. # TP = 73977    # (True Positive)   ：将正类预测为正类数,真实为0,预测为0
132. # FP = 27       # (False Positive)  ：将负类预测为正类数,真实为1,预测为0
133. # .ravel()就是把矩阵拉成一维的
134. TN, FP, FN, TP = confusion\_matrix(y\_true=self.y\_test, y\_pred=self.y\_pred).ravel()
135. self.plot\_confusion\_matrix(self.cm, self.labelencoder\_y.classes\_)
136. **print**("正类预测的准确率: %f" % (TP\*1.0 / (TP + FN)))
137. **print**("负类预测的准确率: %f" % (TN\*1.0 / (TN + FP)))
138. correct\_predictions = TN + TP
139. total\_predictions = TN + FN + TP + FP
140. **print**("总的准确率(公式): %f" % (correct\_predictions\*1.0 / total\_predictions))
142. **print**("总的准确率(机器): %f" % accuracy\_score(self.y\_test, self.y\_pred))
144. **def** plot\_grid\_search(self):
145. self.forest = RandomForestClassifier(random\_state=1)
146. self.param\_grid = {
147. 'n\_estimators': [5, 10],
148. 'max\_depth': [3, None],
149. 'max\_features': [1, 3, 5],
150. 'min\_samples\_split': [2, 5],
151. 'min\_samples\_leaf': [2, 5],
152. 'bootstrap': [True, False],
153. 'criterion': ['gini', 'entropy']
154. }
155. # 模型调参利器 gridSearchCV（网格搜索）
156. self.grid = GridSearchCV(estimator=self.forest, param\_grid=self.param\_grid, cv=2)
157. self.grid.fit(self.x\_train, self.y\_train)
158. **print** (self.grid.best\_estimator\_)
159. **print** ("\nBest Score: ", self.grid.best\_score\_)
161. self.grid.score(self.x\_test, self.y\_test)
163. # threshold：阈值
164. # 针对每一个特征值，把散点图画出来，并把阈值画出来
165. **def** feature\_plot\_array(self, x, i, threshold):
166. base = np.arange(len(x))
167. plt.grid(True)
168. cmap = LinearSegmentedColormap.from\_list('custom blue',
169. [(0, '#ff0000'),
170. (1, '#00ff00')], N=256)
171. # plt.plot(base, x, c='blue')
172. # 画散点图
173. plt.scatter(base, x, c=self.feature\_y, cmap=cmap, marker='.')
174. # plt.title(param\_array[i][0] + " - " + param\_array[i][1])
175. plt.title(self.feature\_param\_array[i][1])
176. # plot threshold 把阈值画出来
177. plt.axhline(y=threshold, color='blue')
178. # plt.legend()函数，给图像加上图例。
179. blue\_plus = mlines.Line2D([], [], color='#00ff00', marker='.', linestyle='None',   markersize=10, label='Clean Host')
180. purple\_plus = mlines.Line2D([], [], color='#ff0000', marker='.', linestyle='None',   markersize=10, label='Infected Host')
181. threshold\_plus = mlines.Line2D([0, 1], [0, 1], color='blue', marker='\_', linestyle='solid', markersize=10, label='Threshold')
182. plt.legend(loc='upper left', handles=[blue\_plus, purple\_plus, threshold\_plus], shadow=True, fontsize='large', fancybox=True, frameon=True)
183. # labels，设置坐标轴字体
184. plt.xlabel('DNS fingerprint Index')
185. # plt.ylabel(param\_array[i][1])
186. # xmin：x轴上的最小值；xmax：x轴上的最大值
187. # xlim设置x轴的数值显示范围。
188. plt.xlim([0, len(x) - 1])
189. plt.show()
191. **def** feature\_plot(self):
193. self.feature\_x = self.df.loc[:, 'P1':'P15'].values
194. self.feature\_y = self.df.Result.str.strip()
195. **from** sklearn.preprocessing **import** LabelEncoder, OneHotEncoder
196. labelencoder\_y = LabelEncoder()
197. self.feature\_y = self.labelencoder\_y.fit\_transform(self.feature\_y)
198. self.feature\_param\_array = [
199. ['P1', 'Nbr. of DNS requests per hour ', 7500],
200. ['P2', 'Nbr. of Distinct DNS requests ', 1500],
201. ['P3', 'Highest Nbr. of requests(single domain)', 1000],
202. ['P4', 'Average Nbr. of requests ', 300],
203. ['P5', 'Highest Nbr. of requests ', 500],
204. ['P6', 'Nbr. of MX Record Queries ', 10],
205. ['P7', 'Nbr. of PTR Record Queries ', 500],
206. ['P8', 'Nbr. of Distinct DNS Servers ', 5],
207. ['P9', 'Nbr. of Distinct TLD  Queried ', 25],
208. ['P10', 'Nbr. of Distinct SLD  Queried ', 500],
209. ['P11', 'Uniqueness ratio ', 500],
210. ['P12', 'Nbr. of Failed Queries', 12],
211. ['P13', 'Nbr. of Distinct Cities', 70],
212. ['P14', 'Nbr. of Distinct Countries ', 30],
213. ['P15', 'Flux ratio per hour ', 100]]
214. **print**("\tMax\t\t|\t\tMin\t\t|\t\tAverage")
215. # 就是range 0-14
216. **for** i **in** np.arange(15):
217. **print**("%-20s%-15s%-s" % (str(max(self.feature\_x[:, i])) ,str(min(self.feature\_x[:, i])) ,str(np.mean(self.feature\_x[:, i]))))
218. fig = plt.gcf()
219. fig.set\_size\_inches(5, 4)
220. **for** i **in** np.arange(15):
221. self.feature\_plot\_array(self.feature\_x[:, i], i, self.feature\_param\_array[i][2])
222. **if** \_\_name\_\_ == '\_\_main\_\_':
223. obj = BotNetworkDetector()
224. obj.import\_db()
225. obj.encode\_values()
226. obj.display\_count()
227. obj.split\_data()
228. obj.feature\_plot()
229. obj.create\_forest()
230. obj.show\_feature\_imp()
231. obj.plot\_predicted\_model()
232. obj.plot\_grid\_search()

#### 3.4.4 特征重要度分析

上述函数show\_feature\_imp()输出了每个特征的重要程度如下：



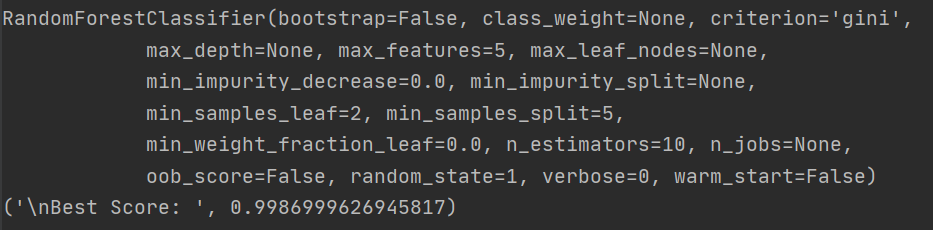
通过排序并画图得到如下的柱状图：



而随机森林算法可以将特征的importance作为模型的输入，因而考虑特征的权值，能够更好的进行DNS流量的分类。

#### 3.4.5 模型调优

算法中的plot\_grid\_search()函数即寻找最佳的适合本项目的随机森林分类器参数，输出的最佳参数和最高准确率如下：



#### 3.4.6 准确率

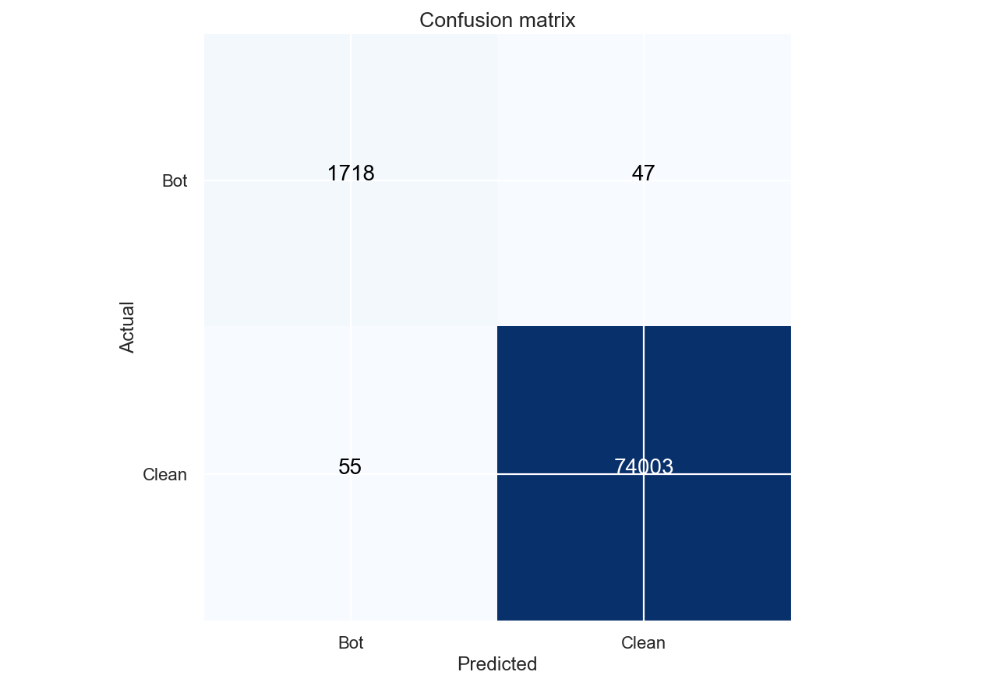
###### 3.4.6.1 混淆矩阵

首先要了解准确率的概念：一个二分类的模型，其预测结果有四类：

* 将负类预测为负类数,真实为Bot，预测为Bot
* 将正类预测为负类数,真实为Clean，预测为Bot
* 将正类预测为正类数,真实为Clean，预测为Clean
* 将负类预测为正类数,真实为Bot，预测为Clean

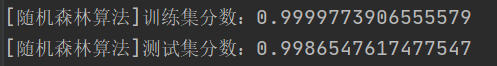
因此混淆矩阵即为这四项，输出和图像表示如下：





###### 3.4.6.2 准确率

（1）机器学习模型的准确率：



（2）引入混淆矩阵后的准确率输出如下



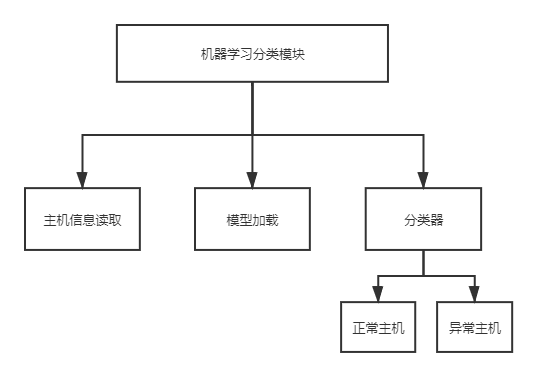
#### 3.4.7 模型保存

sklearn库可以将训练好的模型保存，待下次使用的同时直接加载即可，可以提高检测速度，不用每次都生成一次机器学习的结果，算法如下：

1. **def** save\_model(self):
2. with open('model', 'wb') as f:
3. cPickle.dump(self.rf\_classifier, f)
5. **def** load\_model(self):
6. with open('model', 'rb') as f:
7. self.\_rf\_classifier = cPickle.load(f)

### 3.5 机器学习分类模块

#### 3.5.1 模块功能描述



本模块的主要功能为利用机器学习的结果，以生成好的特征值文件作为输入，输出分类好的僵尸主机或正常主机。

#### 3.5.2 算法

（1）主函数里判断是实时检测还是文件检测，从而转换到不同的模式

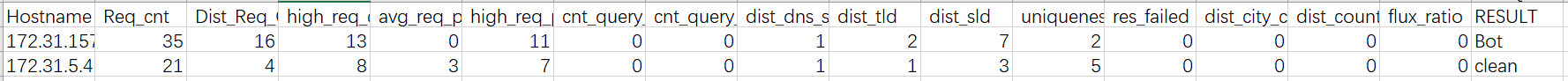
1. # 机器学习分类
2. fn\_in = "../Output/DNS\_FP.csv"
3. fn\_out = "../Output/DNS\_FP\_RESULT.csv"
5. df = pd.read\_csv(fn\_in, low\_memory=False)
6. **if** df.shape[0] > 0:
7. obj = BotNetDetector()
8. obj.load\_model()
9. obj.get\_x\_data(fn\_in)
10. obj.get\_y\_data(fn\_out)
12. **if** self.cmd == 1:
13. result = []
14. self.get\_bot\_hosts(result)
15. **if** len(result) > 0:
16. df\_in = pd.read\_csv(fn\_in, low\_memory=False)
17. df\_out = pd.read\_csv(fn\_out, low\_memory=False)
18. df\_in.to\_csv("../Output/DNS\_FP\_BOT.csv")
19. df\_out.to\_csv("../Output/DNS\_FP\_RESULT\_BOT.csv")
20. **print**("[!]Bot Network Detected！！")
21. **print**("[!]Hosts：")
22. i = 1
23. **print**("序号\t\t\tIP\t\t请求数")
24. **for** m\_host **in** result:
25. **print** (str(i) + '.\t' + m\_host + '\t' + str(self.hosts[m\_host].req\_count))
26. i += 1
27. **print**("\n继续监测中...(输入exit退出)")

（2）由于已经训练好模型，直接加载即可，随后获取x和y进行分类输出即可，对于的子函数如下：

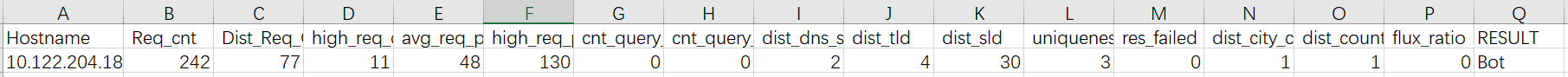
1. **def** load\_model(self):
2. with open('model', 'rb') as f:
3. self.\_rf\_classifier = cPickle.load(f)
5. **def** get\_x\_data(self, filename):
6. self.\_df\_ = pd.read\_csv(filename, low\_memory=False)
7. # print("测试集中一共有%s行，%s 列" % self.\_df\_.shape)
8. self.x\_value = self.\_df\_[self.\_df\_.columns.difference(['Hostname'])].values
9. **def** get\_y\_data(self, filename):
10. self.y\_value = self.\_rf\_classifier.predict(self.x\_value)
11. result = []
12. **for** item **in** self.y\_value:
13. **if** item == 1:
14. result.append("clean")
15. **else**:
16. result.append("Bot")
17. self.\_df\_["RESULT"] = result
18. self.\_df\_.to\_csv(filename)

#### 3.5.3 输出

（1）所有主机的特征值信息和分类信息

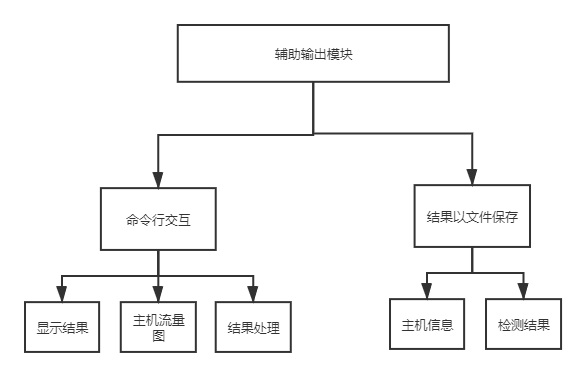


（2）仅包含僵尸主机的特征值信息和分类信息



### 3.6 辅助输出模块

#### 3.6.1 模块功能描述



本模块主要是辅助模块，是与用户交互的界面，询问/帮助用户如何使用程序、处理检测的结果。

#### 3.6.2 算法

（1）程序初始阶段

1. **print**("========= Welcome to Bot Network Detector =========")
2. **print**("||           Update Time : 2020-12-3             ||")
3. **print**("||           Version     : 1.0                   ||")
4. **print**("======Program Started at " + datetime.strftime(datetime.now(),
5. '%Y-%m-%d  %H:%M:%S') + "======")
6. **while** True:
7. **print**("[cmd]\n[1]实时流量监控\n[2]pcap文件检测\n[3]退出系统")
8. cmd\_init = input("请输入命令(1-3的某个整数): ")
9. **if** cmd\_init.isdigit():
10. **if** int(cmd\_init) == 1:
11. **print**("正在进行实时dns流量检测(输入exit退出)...")
12. flag = True
13. t\_realtime\_sniff = threading.Thread(target=realtime\_sniff)
14. t\_get\_file = threading.Thread(target=get\_file)
15. t\_cmd\_exit = threading.Thread(target=cmd\_exit)
16. t\_realtime\_detect = threading.Thread(target=realtime\_detect)
17. t\_realtime\_sniff.start()
18. t\_get\_file.start()
19. t\_cmd\_exit.start()
20. t\_realtime\_detect.start()
22. t\_cmd\_exit.join()
23. t\_realtime\_detect.join()
24. t\_get\_file.join()
25. t\_realtime\_sniff.join()
26. **continue**
27. **elif** int(cmd\_init) == 2:
28. **print**("请将待扫描的pcap文件放入文件夹InputFile内(支持多个)")
29. cmd\_infile = input("确认放入后请输入1")
30. **if** int(cmd\_infile) == 1:
31. **try**:
32. folder = "../InputFile/"
33. filename = "dns\_traffic.pcap"
34. obj\_dns\_parser = PcapParser(10000000, 3, folder, filename, 1, 2)
35. obj\_dns\_parser.start\_parse()
36. **except** Exception as e:
37. **print**(e)
38. **continue**
39. **elif** int(cmd\_init) == 3:
40. input("\n欢迎下次使用，bye-bye！")
41. exit(0)
42. **break**
43. **print**("[命令错误]请重新输入，请输入1-3的某个整数")
45. input("\n欢迎下次使用，bye-bye！")
46. exit(0)

（2）扫描检测结束，打开终端交互

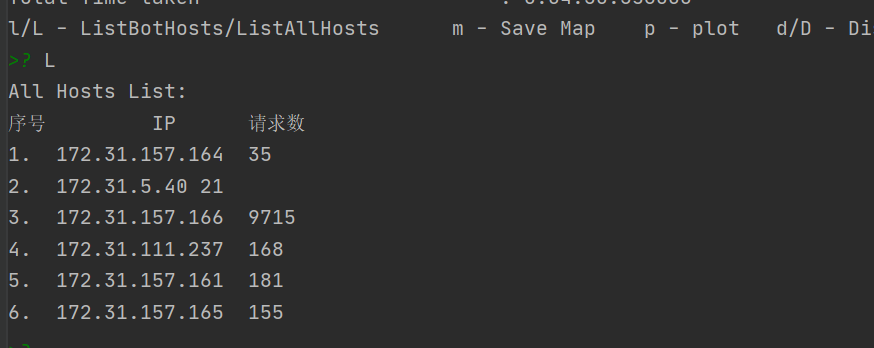
1. **def** start\_console(self):
2. syntax = 'l/L - ListBotHosts/ListAllHosts \t m - Save Map \t p - plot \td/D - Display/Save \t h - saveHtml \t x - saveCSV \t F - Find Req URl\t f – Fin d Resolved IP\t q - quit'
3. **print**(syntax)
4. choice = 'l'
5. **while** choice != 'q':
6. choice = raw\_input()
7. **if** choice == 'L':
8. result = []
9. self.get\_all\_hosts(result)
10. i = 1
11. **print** ('All Hosts List: ')
12. **print**("序号\t\t\tIP\t\t请求数")
13. **for** m\_host **in** result:
14. **print** (str(i) + '.\t' + m\_host + '\t' + str(self.hosts[m\_host].req\_count))
15. i += 1
16. **elif** choice == 'l':
17. result = []
18. self.get\_bot\_hosts(result)
19. i = 1
20. **print** ('Bot Hosts List: ')
21. **if** len(result) > 0:
22. **print**("序号\t\t\tIP\t\t请求数")
23. **for** m\_host **in** result:
24. **print** (str(i) + '.\t' + m\_host + '\t' + str(self.hosts[m\_host].req\_count))
25. i += 1
26. **else**:
27. **print**("没有找到！")
28. **elif** choice == 'p':
29. **print** ('Enter Hostname :')
30. hostname = raw\_input()
31. self.plot(hostname)
32. **elif** choice == 'f':
33. **print** ('Enter Resolved IP :')
34. resolved\_ip = raw\_input()
35. self.find\_resolved\_ip(resolved\_ip)
36. **elif** choice == 'F':
37. **print** ('Enter Request URL :')
38. req\_url = raw\_input()
39. self.find\_req\_url(req\_url)
40. **elif** choice == 'd':
41. **print** ('Enter Hostname :')
42. hostname = raw\_input()
43. self.display\_host(hostname)
44. **elif** choice == 'D':
45. **print** ('Enter Hostname :')
46. hostname = raw\_input()
47. self.save\_host(hostname)
48. **elif** choice == 'm':
49. **print** ('Enter Hostname :')
50. hostname = raw\_input()
51. tmp\_str = ''
52. self.save\_map(hostname, tmp\_str)
53. **print** (tmp\_str)
54. **elif** choice == 'q':
55. **continue**
56. **elif** choice == 'x':
57. self.save\_csv()
58. **elif** choice == 'h':
59. self.save\_html()
60. **else**:
61. **print** ('Invalid Choice !!')
62. **print** (syntax)
63. **print** ('console>')
65. **print** ('Console Terminated.')

（3）下面是详细的子函数部分，包括获取特定ip的信息，查找主机的请求url，输出检测结果，将结果保存为csv、将结果保存为html，获取所有主机/获取僵尸主机的ip等信息：

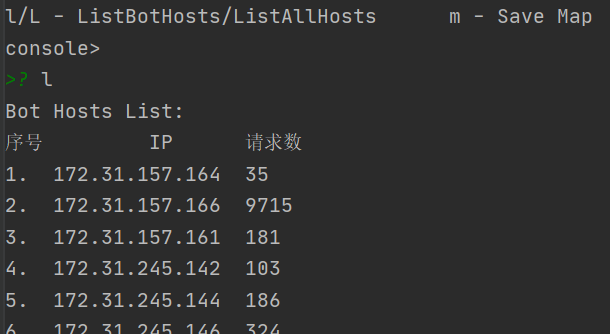
1. **def** find\_resolved\_ip(self, resolved\_ip):
2. **for** m\_hosts **in** self.hosts:
3. obj = self.hosts[m\_hosts]
4. next\_host = False
5. **for** m\_urls **in** obj.domain:
6. token\_list = obj.domain[m\_urls]
7. **for** token **in** token\_list.list:
8. request = token\_list.list[token]
9. **for** res **in** request.response:
10. **if** res.resolved\_ip == resolved\_ip:
11. **print** (m\_hosts)
12. next\_host = True
13. **break**
14. **if** next\_host:
15. **break**
17. **if** next\_host:
18. **break**
19. **if** next\_host:
20. **continue**
22. **def** find\_req\_url(self, req\_url):
23. **for** m\_hosts **in** self.hosts:
24. obj = self.hosts[m\_hosts]
25. next\_host = False
26. **for** m\_urls **in** obj.domain:
27. **if** m\_urls == req\_url:
28. **print** (m\_hosts)
29. next\_host = True
30. **break**
31. **if** next\_host:
32. **continue**
34. **def** summary(self):
35. **print** ('Total Number of Hosts :' + str(len(self.hosts)))
36. **print** ('Number of Hosts with over 300 Different Requests:')
37. i = 1
38. **for** m\_hosts **in** self.hosts:
39. obj = self.hosts[m\_hosts]
40. **if** len(obj.domain) > 300:
41. **print** (m\_hosts + '\t' + str(len(obj.domain)))
43. **def** save\_html(self):
44. tmp\_str = "<html><body><table border='1px'><thead><tr><th colspan='5'>DNS Summary</th></tr></thead>"
45. **print** ('Saving Requests:' + str(len(self.hosts)))
46. host\_cnt = 0
47. **for** m\_hosts **in** self.hosts:
48. host\_cnt += 1
49. tmp\_str += '<tr><td>' + str(host\_cnt) + "</td><td colspan='4'>" + m\_hosts + '</td></tr>'
50. obj = self.hosts[m\_hosts]
51. url\_cnt = 0
52. **for** m\_urls **in** obj.domain:
53. url\_cnt += 1
54. tmp\_str += '<tr><td></td><td>' + str(url\_cnt) + "</td><td colspan='4'>" + m\_urls + '</td></tr>'
55. token\_list = obj.domain[m\_urls]
56. token\_cnt = 0
57. **for** token **in** token\_list.list:
58. token\_cnt += 1
59. request = token\_list.list[token]
60. tmp\_str += '<tr><td></td><td></td><td>' + str(
61. token\_cnt) + '</td><td >' + request.txn\_id + '</td><td >' + request.req\_type + '</td><td >' + request.req\_timestamp + '</td></tr>'
62. res\_cnt = 0
63. **for** res **in** request.response:
64. res\_cnt += 1
65. tmp\_str += '<tr><td></td><td></td><td></td><td>' + str(
66. res\_cnt) + '</td><td >' + res.res\_code + '</td><td >' + res.ttl + '</td><td >' + res.resolved\_ip + '</td><td >' + res.res\_timestamp + '</td></tr>'
68. tmp\_str += '</table></body></html>'
69. xml\_outfile = open(self.filename + '.html', 'w')
70. xml\_outfile.write(tmp\_str)
71. xml\_outfile.close()
73. **def** save\_csv(self):
74. i = 1
75. tmp\_str = 'Hostname,count,nbr\_requests,nbr\_unique\_req,avg\_req\_min,max\_req\_min,failed\_cnt,ratio,nbr\_countries,req\_type,sum\_url,sum\_token\n'
76. **for** m\_hosts **in** self.hosts:
77. tmp\_str += m\_hosts + ','
78. obj = self.hosts[m\_hosts]
79. tmp\_str += str(obj.req\_count) + ','
80. tmp\_str += str(obj.nbr\_of\_requests) + ','
81. tmp\_str += str(obj.nbr\_of\_distinct\_requests) + ','
82. tmp\_str += str(obj.avg\_req\_per\_min) + ','
83. tmp\_str += str(obj.max\_req\_per\_min) + ','
84. tmp\_str += str(obj.failed\_req\_count) + ','
85. tmp\_str += str(obj.ratio\_req\_response) + ','
86. tmp\_str += str(obj.nbr\_of\_countries) + ','
87. tmp\_str += str(obj.req\_type) + ','
88. tmp\_str += str(obj.sum\_url\_len) + ','
89. tmp\_str += str(obj.sum\_nbr\_domain\_token) + '\n'
91. xml\_outfile = open('output/' + self.filename + '.csv', 'w')
92. xml\_outfile.write(tmp\_str)
93. xml\_outfile.close()
95. **def** get\_all\_hosts(self, result):
96. df = pd.read\_csv("../Output/DNS\_FP\_RESULT.csv", usecols=[1, 17])
97. **for** i **in** range(df.shape[0]):
98. hostname = df['Hostname'][i][:-7]
99. result.append(hostname)
101. **def** get\_bot\_hosts(self, result):
102. df = pd.read\_csv("../Output/DNS\_FP\_RESULT.csv", usecols=[1, 17])
103. **for** i **in** range(df.shape[0]):
104. hostname = df['Hostname'][i][:-7]
105. \_result = df['RESULT'][i]
106. **if** \_result == 'Bot':
107. result.append(hostname)

#### 3.6.3 输入输出

（1）扫描结束，输入L，列出所有主机信息：

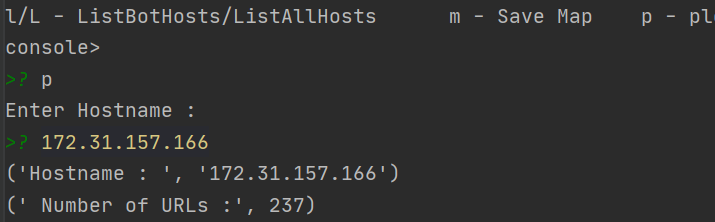


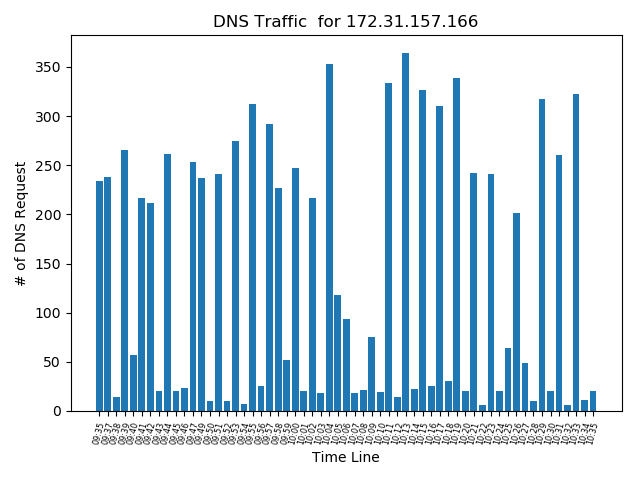
（2）输入l，列出所有感染了僵尸网络的主机信息：



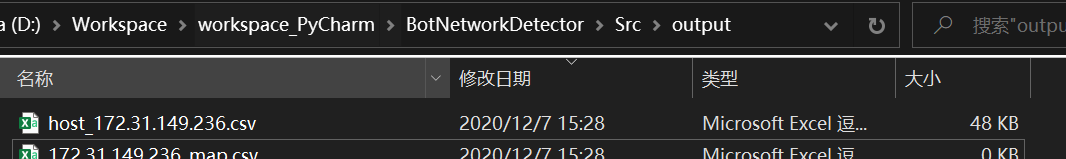
（3）输入p命令，输入主机ip=172.31.157.166，画出该主机的流量图：

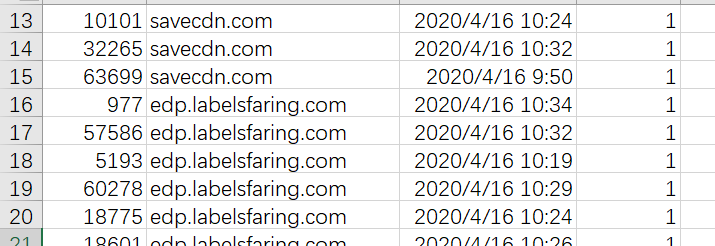
当对某一特定的IP的流量进行审查的时候，有时候仅仅观察数据包或者一些数字信息是无法对其流量特点有比较直观的认识的。尤其是僵尸网络有着在短时间内查询数量高爆发的特点，正常主机的流量在时间轴上比较均匀，这一点只有通过网络流量与实践的关系图才能比较直观的看出特定主机的流量特点，那么为了更加直观的观察一个主机的DNS流量与时间的关系，绘制出这个主机的流量在时间轴上的分布图像就是一个很必要的功能了。





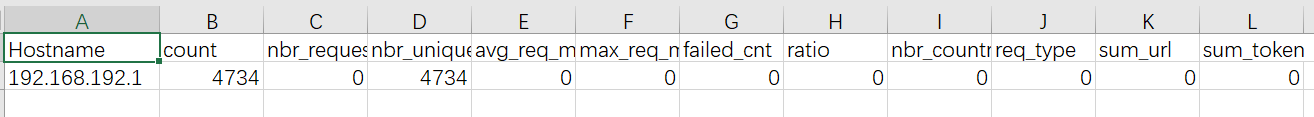
（4）输入D和主机ip=172.31.157.166，可以看到在output文件夹内生成了该主机的信息：





（5）输入x保存为csv，保存本次扫描的结果





（6）输入h保存为html，即保存DNS扫描的结果（Pcap文件的摘要）

