

# DNS实验报告

姓名: 郑雨霏

学号: PB19050946

## 实验环境

实验设备: MacBook Pro

实验环境: macOS 11.4 Visual Studio Code 1.16.2

## 解释程序

先利用套接字获得消息, 利用handle函数处理报文, handle函数中调用DNS\_Packege函数处理数据, 判断报文类型, 如果在example.txt中, 则利用generate\_response函数获得相应响应报文, 转发回去, 否则转发出去, 再接收响应报文转发回去

```
1  #!/usr/bin/python3
2  # -*- coding: utf-8 -*-
3  import socket
4  import threading
5  from time import time
6
7
8  class DNS_Relay_Server:      #一个relay server实例, 通过缓存文件和外部地址来初始化
9      def __init__(self,cache_file,name_server):
10         #url_IP字典:通过域名查询ID
11         self.url_ip = {}
12         self.cache_file = cache_file
13         self.load_file()
14         self.name_server = name_server
15         #trans字典: 通过DNS响应的ID来获得原始的DNS数据包发送方
16         self.trans = {}
17
18         #加载文件
19     def load_file(self,):
20         f = open(self.cache_file,'r',encoding='utf-8')#通过读的方式打开文件
21         for line in f:
22             ip,name = line.split(' ')#利用空格将ip和名字隔开
23             self.url_ip[name.strip('\n')] = ip#用换行符将name隔开, 并与ip对应放入字典
```

中

```

24         f.close()#关闭文件
25
26     #运行dns
27     def run(self):
28         buffer_size = 512#将读取的大小设置为512
29         #socket配置和端口关联
30         server_socket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
31         server_socket.bind(('',53))
32         server_socket.setblocking(False)
33         #不断读取信息并利用handle函数进行处理
34         while True:
35             try:
36                 data,addr = server_socket.recvfrom(buffer_size)
37                 threading.Thread(target=self.handle,args=
38 (server_socket,data,addr)).start()
39             except:
40                 continue
41
42     #处理dns报文
43     def handle(self,server_socket,data,addr):
44         #调用DNS_Package类来解析data里面的数据，如报文id、类型、query中的名字等
45         RecvDp = DNS_Packege(data)
46         id = RecvDp.ID
47         name = RecvDp.name;
48         #如果是请求报文，判断名字是否在example.txt文件中，即字典url_ip中，且是一个请求报文，
49         #若存在则本地生产应答报文，否则将消息转发出去
50         print(name)#输出获得的报文中的名字，用于debug
51         if not RecvDp.QR :
52             if name in self.url_ip and RecvDp.type == 1:
53                 #如果是个设置为无效的ip，则将Intercepted标记为1，否则为0
54                 if self.url_ip[name] == '0.0.0.0':
55                     Intercepted = 1
56                 else :
57                     Intercepted = 0
58                 #调用generate_response函数生成应答报文，将name对应的ip传入函数中
59                 respond =
60 RecvDp.generate_response(self.url_ip[name],Intercepted)
61                 #发送回请求报文中给定的域名服务器和端口
62                 server_socket.sendto(respond,addr)
63             else:
64                 #否则将报文转发出去，并记录下请求报文中的域名服务器和查询的名字
65                 server_socket.sendto(data, self.name_server)
66                 self.trans[id] = (addr, name)
67             #statement
68             #如果不是请求报文，则转发回原来存在trans中的地址，转发完成后删除
69             if RecvDp.QR :
70                 if id in self.trans:
71                     target_addr, name = self.trans[id]
72                     server_socket.sendto(data, target_addr)
73                     del self.trans[id]
74             #statement

```

```

73 #解析报文
74 class DNS_Packege:          #一个DNS Frame实例，用于解析和生成DNS帧
75     def __init__(self,data):
76         Msg_arr = bytearray(data)
77         #ID
78         self.ID = (Msg_arr[0] << 8 ) +Msg_arr[1]
79         # FLAGS
80         self.QR = Msg_arr[2] >> 7
81         # 资源记录数量
82         self.QDCOUNT = (Msg_arr[4] << 8) + Msg_arr[5]
83         #self.ANSWER = (Msg_arr[6] << 8) + Msg_arr[7]
84         self.AUTHOR = (Msg_arr[8] << 8 ) + Msg_arr[9]
85         self.ADDI = (Msg_arr[10] << 8 ) + Msg_arr[11]
86         #query内容解析
87         """data -> name, querybytes, type, classify, len"""
88         self.name = []
89         self.name_length = 0
90         name_block = int(Msg_arr[12])
91         part = ''
92         #通过实验文档中所给的格式（长度+字符+长度+字符+.....+\x0）解析name
93         while(name_block != 0):
94             i = 1;
95             if( self.name_length != 0 ):
96                 self.name.append(".")
97             while i <= name_block:
98                 #part = part + (chr(data[12 + self.name_length + i] ))
99                 self.name.append(chr(data[12 + self.name_length + i] ))
100                 i = i + 1
101                 #self.name.append(part)
102                 #part = ''
103                 self.name_length = self.name_length + name_block + 1
104                 name_block = int(Msg_arr[12 + self.name_length])
105         #将list类型转化为str类型，并解析剩下的信息
106         self.name = ''.join(self.name)
107         self.name_length = self.name_length + name_block + 1;
108         self.type = (Msg_arr[12 + self.name_length ] <<8) + Msg_arr[12 +
self.name_length + 1]
109         self.classify = (Msg_arr[12 + self.name_length + 2] <<8) + Msg_arr[12 +
self.name_length + 3];
110         self.len = self.name_length + 4
111         self.data = data
112
113
114
115         #生成回答
116         def generate_response(self,ip,Intercepted):
117             #如果不是无效的ip，则按照格式生成响应报文
118             if not Intercepted:
119                 #初始化res，并设定长度
120                 res = bytearray(32 + self.name_length)
121                 #ID
122                 res[0] = self.ID >> 8

```

```

123         res[1] = self.ID % 256
124         #FLAGS
125         res[2] = 0x81
126         res[3] = 0x80
127         # 资源记录数量
128         res[4] = self.QDCOUNT >> 8
129         res[5] = self.QDCOUNT % 256
130         res[6] = 0x00
131         res[7] = 0x01
132         res[8] = self.AUTHOR >> 8
133         res[9] = self.AUTHOR % 256
134         res[10] = self.ADDI >> 8
135         res[11] = self.ADDI % 256
136         #query内容解析
137         for i in range(12, 16 + self.name_length):
138             res[i] = self.data[i]
139         #使用偏移指针代替重复的字符串，域名偏移量固定12字节
140         res[16 + self.name_length] = 0xc0
141         res[17 + self.name_length] = 0x0c
142         #自定义FLAGS、资源记录数量等
143         res[18 + self.name_length] = 0x00
144         res[19 + self.name_length] = 0x01
145         res[20 + self.name_length] = 0x00
146         res[21 + self.name_length] = 0x01
147         res[22 + self.name_length] = 0x00
148         res[23 + self.name_length] = 0x00
149         res[24 + self.name_length] = 0x0d
150         res[25 + self.name_length] = 0x34
151         res[26 + self.name_length] = 0x00
152         res[27 + self.name_length] = 0x04
153         #利用.将ip分割为四个字段，转化为int型，存入res中，返回bytes型
154         ip_ = ip.split('.')
155         ip_1 = int(ip_[0])
156         ip_2 = int(ip_[1])
157         ip_3 = int(ip_[2])
158         ip_4 = int(ip_[3])
159         res[28 + self.name_length] = ip_1
160         res[29 + self.name_length] = ip_2
161         res[30 + self.name_length] = ip_3
162         res[31 + self.name_length] = ip_4
163         return bytes(res)
164     else:
165         #如果是无效的ip地址，则将前一种情况的flags更改为0x8583，指示名字错误，
166         res = bytearray(32 + self.name_length)
167         res[0] = self.ID >> 8
168         res[1] = self.ID % 256
169         res[2] = 0x85
170         res[3] = 0x83
171         res[4] = self.QDCOUNT >> 8
172         res[5] = self.QDCOUNT % 256
173         res[6] = 0x00
174         res[7] = 0x01

```

```

175         res[8] = self.AUTHOR >> 8
176         res[9] = self.AUTHOR % 256
177         res[10] = self.ADDI >> 8
178         res[11] = self.ADDI % 256
179         for i in range(12, 16 + self.name_length):
180             res[i] = self.data[i]
181         res[16 + self.name_length] = 0xc0
182         res[17 + self.name_length] = 0x0c
183         res[18 + self.name_length] = 0x00
184         res[19 + self.name_length] = 0x01
185         res[20 + self.name_length] = 0x00
186         res[21 + self.name_length] = 0x01
187         res[22 + self.name_length] = 0x00
188         res[23 + self.name_length] = 0x00
189         res[24 + self.name_length] = 0x0d
190         res[25 + self.name_length] = 0x34
191         res[26 + self.name_length] = 0x00
192         res[27 + self.name_length] = 0x04
193         ip_ = ip.split('.')
194         ip_1 = int(ip_[0])
195         ip_2 = int(ip_[1])
196         ip_3 = int(ip_[2])
197         ip_4 = int(ip_[3])
198         res[28 + self.name_length] = ip_1
199         res[29 + self.name_length] = ip_2
200         res[30 + self.name_length] = ip_3
201         res[31 + self.name_length] = ip_4
202         return bytes(res)
203
204
205 if __name__ == '__main__':
206     cache_file = 'example.txt'
207     name_server=('223.5.5.5',53)
208     relay_server = DNS_Relay_Server(cache_file,name_server)    #构造一个
DNS_Relay_Server实例
209     relay_server.run() #运行

```

## 在程序中的处理

DNS query的处理：

```

1  #query内容解析
2  """data -> name, querybytes, type, classify, len"""
3  #初始化name为list, name length为0, name block为当前块name的长度
4  self.name = []
5  self.name_length = 0
6  name_block = int(Msg_arr[12])
7  #通过实验文档中所给的格式（长度+字符+长度+字符+.....+\x0）解析name
8  #当没有访问到\x0时，继续向后

```

```

9         while(name_block != 0):
10             #i为记录长度的counter
11             i = 1;
12             #如果当前不是第一part的name, 先加一个".", 再加后面的part名字
13             if( self.name_length != 0 ):
14                 self.name.append(".")
15             #按照长度, 访问该部分name
16             while i <= name_block:
17                 self.name.append(chr(data[12 + self.name_length + i] ))
18                 i = i + 1
19             #记录当前name长度
20             self.name_length = self.name_length + name_block + 1
21             #继续查看下一部分name的长度
22             name_block = int(Msg_arr[12 + self.name_length])
23             #将list类型转化为str类型, 并解析剩下的信息
24             self.name = ''.join(self.name)
25             #记录当前name长度
26             self.name_length = self.name_length + name_block + 1;
27             #按照格式记录type和classify
28             self.type = (Msg_arr[12 + self.name_length ] <<8) + Msg_arr[12 +
self.name_length + 1]
29             self.classify = (Msg_arr[12 + self.name_length + 2] <<8) + Msg_arr[12 +
self.name_length + 3];
30             #记录query部分总长度
31             self.len = self.name_length + 4
32             #记录总的的数据
33             self.data = data

```

DNS响应消息处理:

```

1 def generate_response(self,ip,Intercepted):
2     #如果不是无效的ip, 则按照格式生成响应报文
3     if not Intercepted:
4         #初始化res, 并设定长度
5         res = bytearray(32 + self.name_length)
6         #ID
7         res[0] = self.ID >> 8
8         res[1] = self.ID % 256
9         #FLAGS
10        res[2] = 0x81
11        res[3] = 0x80
12        # 资源记录数量
13        res[4] = self.QDCOUNT >> 8
14        res[5] = self.QDCOUNT % 256
15        # 回答记录数自定义为1
16        res[6] = 0x00
17        res[7] = 0x01
18        res[8] = self.AUTHOR >> 8
19        res[9] = self.AUTHOR % 256
20        res[10] = self.ADDI >> 8
21        res[11] = self.ADDI % 256
22        #query内容解析

```

```

23         for i in range(12, 16 + self.name_length):
24             res[i] = self.data[i]
25         #使用偏移指针代替重复的字符串, 域名偏移量固定12字节
26         res[16 + self.name_length] = 0xc0
27         res[17 + self.name_length] = 0x0c
28         #自定义FLAGS、资源记录数量等
29         res[18 + self.name_length] = 0x00
30         res[19 + self.name_length] = 0x01
31         res[20 + self.name_length] = 0x00
32         res[21 + self.name_length] = 0x01
33         res[22 + self.name_length] = 0x00
34         res[23 + self.name_length] = 0x00
35         res[24 + self.name_length] = 0x0d
36         res[25 + self.name_length] = 0x34
37         res[26 + self.name_length] = 0x00
38         res[27 + self.name_length] = 0x04
39         #利用.将ip分割为四个字段, 转化为int型, 存入res中, 返回bytes型
40         ip_ = ip.split('.')
41         ip_1 = int(ip_[0])
42         ip_2 = int(ip_[1])
43         ip_3 = int(ip_[2])
44         ip_4 = int(ip_[3])
45         res[28 + self.name_length] = ip_1
46         res[29 + self.name_length] = ip_2
47         res[30 + self.name_length] = ip_3
48         res[31 + self.name_length] = ip_4
49         return bytes(res)
50     else:
51         #如果是无效的ip地址, 则将前一种情况的flags更改为0x8583, 指示名字错误
52         res = bytearray(32 + self.name_length)
53         res[0] = self.ID >> 8
54         res[1] = self.ID % 256
55         res[2] = 0x85
56         res[3] = 0x83
57         res[4] = self.QDCOUNT >> 8
58         res[5] = self.QDCOUNT % 256
59         res[6] = 0x00
60         res[7] = 0x01
61         res[8] = self.AUTHOR >> 8
62         res[9] = self.AUTHOR % 256
63         res[10] = self.ADDI >> 8
64         res[11] = self.ADDI % 256
65         for i in range(12, 16 + self.name_length):
66             res[i] = self.data[i]
67         res[16 + self.name_length] = 0xc0
68         res[17 + self.name_length] = 0x0c
69         res[18 + self.name_length] = 0x00
70         res[19 + self.name_length] = 0x01
71         res[20 + self.name_length] = 0x00
72         res[21 + self.name_length] = 0x01
73         res[22 + self.name_length] = 0x00
74         res[23 + self.name_length] = 0x00

```

```

75         res[24 + self.name_length] = 0x0d
76         res[25 + self.name_length] = 0x34
77         res[26 + self.name_length] = 0x00
78         res[27 + self.name_length] = 0x04
79         ip_ = ip.split('.')
80         ip_1 = int(ip_[0])
81         ip_2 = int(ip_[1])
82         ip_3 = int(ip_[2])
83         ip_4 = int(ip_[3])
84         res[28 + self.name_length] = ip_1
85         res[29 + self.name_length] = ip_2
86         res[30 + self.name_length] = ip_3
87         res[31 + self.name_length] = ip_4
88         return bytes(res)

```

## 程序的执行和输出

配置dns后的浏览器：无法显示知乎中的图片



程序运行

182.61.200.7 [www.baidu.com](http://www.baidu.com)

127.0.0.1 [www.test1.com](http://www.test1.com)

可以直接通过程序创建响应报文返回

[www.bilibili.com](http://www.bilibili.com)需要将报文转发出去后再转发给原地址



pic1.zhimg.com、pic2.zhimg.com为0.0.0.0，需要响应名字错误报文

```
+ ~ nslookup -ty=A www.baidu.com 127.0.0.1
Server:      127.0.0.1
Address:     127.0.0.1#53

Non-authoritative answer:
Name: www.baidu.com
Address: 182.61.200.7

+ ~ nslookup -ty=A www.bilibili.com 127.0.0.1
Server:      127.0.0.1
Address:     127.0.0.1#53

Non-authoritative answer:
www.bilibili.com canonical name = s.w.bilicdn1.com.
Name: s.w.bilicdn1.com
Address: 119.3.65.116
Name: s.w.bilicdn1.com
Address: 119.3.74.154
Name: s.w.bilicdn1.com
Address: 120.92.147.239
Name: s.w.bilicdn1.com
Address: 119.3.32.96
Name: s.w.bilicdn1.com
Address: 119.3.65.164
Name: s.w.bilicdn1.com
Address: 120.92.168.13
Name: s.w.bilicdn1.com
Address: 119.3.77.172
Name: s.w.bilicdn1.com
Address: 119.3.44.211
Name: s.w.bilicdn1.com
Address: 119.3.33.86
Name: s.w.bilicdn1.com
Address: 119.3.70.188

+ ~ nslookup -ty=A pic1.zhimg.com 127.0.0.1
Server:      127.0.0.1
Address:     127.0.0.1#53

** server can't find pic1.zhimg.com: NXDOMAIN

+ ~ nslookup -ty=A www.test1.com 127.0.0.1
Server:      127.0.0.1
Address:     127.0.0.1#53

Non-authoritative answer:
Name: www.test1.com
```

```
DNS_Relay.py x
Users > zhengyufei > Desktop > study > network > lab1 > DNS_Relay.py > DNS_Package > generate_response

140         res[30 + self.name_length] = ip_3
141         res[31 + self.name_length] = ip_4
142         return bytes(res)
143     else:
144         res = bytearray(32 + self.name_length)
145         res[0] = self.ID >> 8
146         res[1] = self.ID % 256
147         res[2] = 0x85
148         res[3] = 0x83
149         res[4] = self.QDCOUNT >> 8
150         res[5] = self.QDCOUNT % 256
151         res[6] = 0x00
152         res[7] = 0x01
153         res[8] = self.AUTHOR >> 8
154         res[9] = self.AUTHOR % 256
155         res[10] = self.ADDI >> 8
156         res[11] = self.ADDI % 256
157         for i in range(12, 16 + self.name_length):
158             res[i] = self.data[i]
159         res[16 + self.name_length] = 0xc0

问题 输出 调试控制台 终端
Code - lab1 + -  Python 3.9.7 64-bit 0 0 0 Live Share 行 157, 列 7 空格: 4 UTF-8 CRLF Python
```

```
Server:      127.0.0.1
Address:     127.0.0.1#53

Non-authoritative answer:
www.bilibili.com canonical name = s.w.bilicdn1.com.
Name: s.w.bilicdn1.com
Address: 119.3.65.116
Name: s.w.bilicdn1.com
Address: 119.3.74.154
Name: s.w.bilicdn1.com
Address: 120.92.147.239
Name: s.w.bilicdn1.com
Address: 119.3.32.96
Name: s.w.bilicdn1.com
Address: 119.3.65.164
Name: s.w.bilicdn1.com
Address: 120.92.168.13
Name: s.w.bilicdn1.com
Address: 119.3.77.172
Name: s.w.bilicdn1.com
Address: 119.3.44.211
Name: s.w.bilicdn1.com
Address: 119.3.33.86
Name: s.w.bilicdn1.com
Address: 119.3.70.188

+ ~ nslookup -ty=A pic1.zhimg.com 127.0.0.1
Server:      127.0.0.1
Address:     127.0.0.1#53

** server can't find pic1.zhimg.com: NXDOMAIN

+ ~ nslookup -ty=A www.test1.com 127.0.0.1
Server:      127.0.0.1
Address:     127.0.0.1#53

Non-authoritative answer:
Name: www.test1.com
Address: 127.0.0.1

+ ~ nslookup -ty=A pic2.zhimg.com 127.0.0.1
Server:      127.0.0.1
Address:     127.0.0.1#53

** server can't find pic2.zhimg.com: NXDOMAIN

+ ~
```

```
DNS_Relay.py x
Users > zhengyufei > Desktop > study > network > lab1 > DNS_Relay.py > DNS_Package > generate_response

140         res[30 + self.name_length] = ip_3
141         res[31 + self.name_length] = ip_4
142         return bytes(res)
143     else:
144         res = bytearray(32 + self.name_length)
145         res[0] = self.ID >> 8
146         res[1] = self.ID % 256
147         res[2] = 0x85
148         res[3] = 0x83
149         res[4] = self.QDCOUNT >> 8
150         res[5] = self.QDCOUNT % 256
151         res[6] = 0x00
152         res[7] = 0x01
153         res[8] = self.AUTHOR >> 8
154         res[9] = self.AUTHOR % 256
155         res[10] = self.ADDI >> 8
156         res[11] = self.ADDI % 256
157         for i in range(12, 16 + self.name_length):
158             res[i] = self.data[i]
159         res[16 + self.name_length] = 0xc0

问题 输出 调试控制台 终端
Code - lab1 + -  Python 3.9.7 64-bit 0 0 0 Live Share 行 157, 列 7 空格: 4 UTF-8 CRLF Python
```